DOCUMENT RESUME

ED 345 690 IR 015 496

AUTHOR	Moline, Judi, Ed.; And Others
TITLE	Proceedings of the Hypertext Standardization Workshop (Gaithersburg, Maryland, January 16-18, 1990).
INSTITUTION	National Inst. of Standards and Technology, Gaithersburg, MD.
REPORT NO	NIST-SP-500-178
PUB DATE	Mar 90
NOTE	262p.
PUB TYPE	Collected Works - Conference Proceedings (021)
EDRS PRICE	MF01/PC11 Plus Postage.
DESCRIPTORS	*Computer Software; Databases; Higher Education; *Hypermedia; Information Management; Information Systems; Models; *Standards
IDENTIFIERS	National Institute of Standards and Technology

ABSTRACT

This report constitutes the proceedings of a three day workshop on Hypertext Standardization held at the National Institute of Standards and Technology (NIST) on January 16-18, 1990. Efforts towards standardization of hypertext have already been initiated in various interested organizations. The major purpose of the workshop was to provide a forum for presentation and discussion of existing and proposed approaches to standardization, and to consider hypertext system definitions, to identify viable approaches for pursuing standards, to seek commonality among alternatives whenever possible, and to make progress towards a coordinated plan for standards development, i.e., a hypertext reference model. The workshop included plenary sessions and three discussion groups. This proceedings includes 14 papers that were selected for presentation in plenary sessions, reports of the discussion groups, and supplementary materials. Major conclusions of the workshop were that the discussion groups should continue their technical efforts, and that NIST should sponsor at least one more workshop to provide a forum for public discussion of progress. References are included throughout the document. (Author/DB)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED345690

NIST Special Publication 500-178

IR

Computer
Systems
Technology

U.S. DEPARTMENT OF
COMMERCE
National Institute of
Standards and
Technology

NIST

Proceedings of the Hypertext Standardization Workshop January 16–18, 1990 National Institute of Standards and Technology

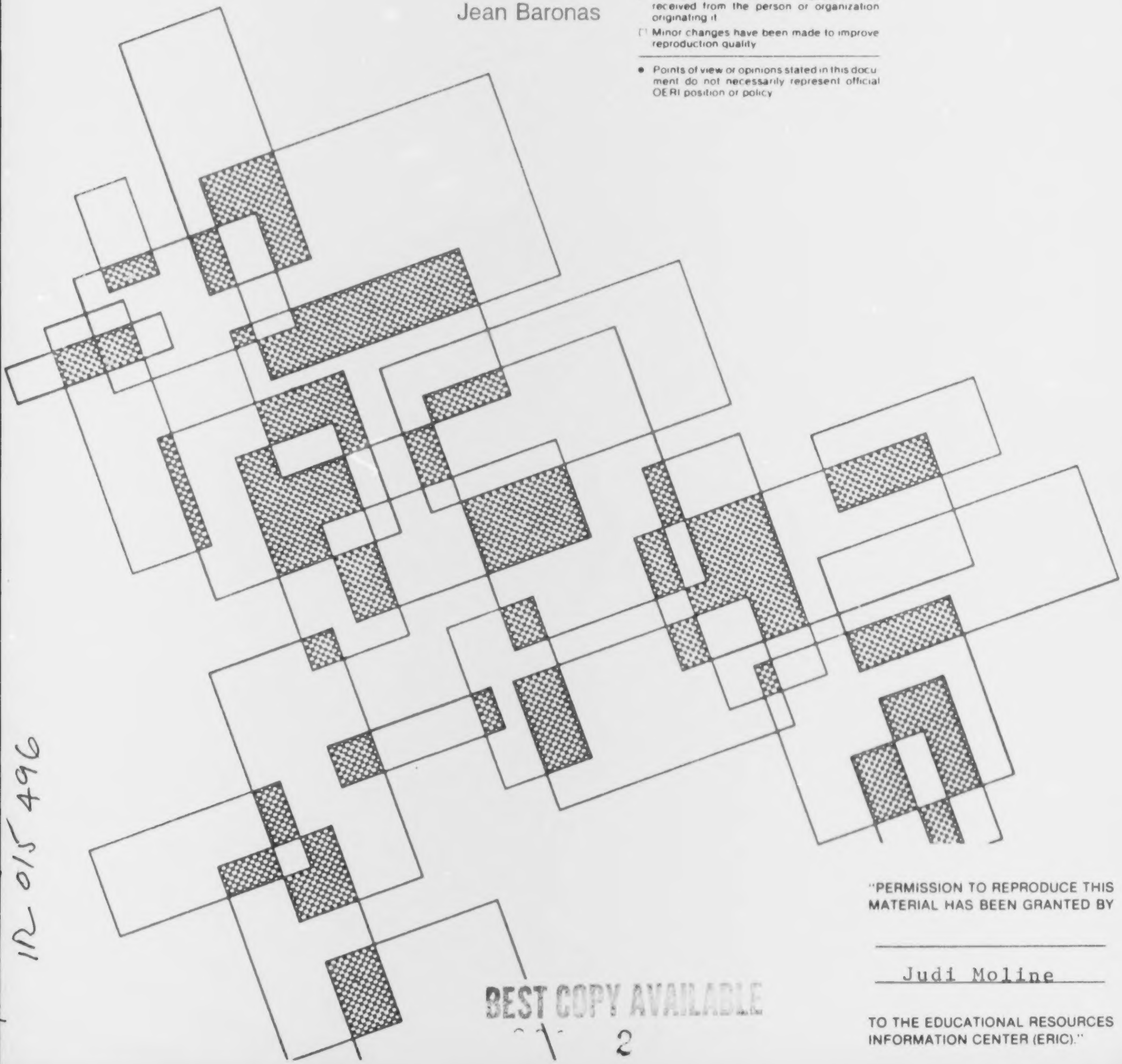
Judi Moline
Dan Benigni
Jean Baronas

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION
CENTER (ERIC)

☐ This document has been reproduced as received from the person or organization originating it

☐ Minor changes have been made to improve reproduction quality

- Points of view or opinions stated in this document do not necessarily represent official OERI position or policy



"PERMISSION TO REPRODUCE THIS
MATERIAL HAS BEEN GRANTED BY

Judi Moline

TO THE EDUCATIONAL RESOURCES
INFORMATION CENTER (ERIC)."

BEST COPY AVAILABLE

IR 015 496

NIST Special Publication 500-178

Proceedings of the Hypertext Standardization Workshop January 16–18, 1990 National Institute of Standards and Technology

Judi Moline, Dan Benigni, and Jean Baronas, Editors

Hypertext Competence Project
National Computer Systems Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

March 1990

U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher, Secretary
NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
John W. Lyons, Director

Reports on Computer Systems Technology

The National Institute of Standards and Technology (NIST) (formerly the National Bureau of Standards) has a unique responsibility for computer systems technology within the Federal government. NIST's National Computer Systems Laboratory (NCSL) develops standards and guidelines, provides technical assistance, and conducts research for computers and related telecommunications systems to achieve more effective utilization of Federal information technology resources. NCSL's responsibilities include development of technical, management, physical, and administrative standards and guidelines for the cost-effective security and privacy of sensitive unclassified information processed in Federal computers. NCSL assists agencies in developing security plans and in improving computer security awareness training. This Special Publication 500 series reports NCSL research and guidelines to Federal agencies as well as to organizations in industry, government, and academia.

National Institute of Standards and Technology Special Publication 500-178
Natl. Inst. Stand. Technol. Spec. Publ. 500-178, 259 pages (Mar. 1990)
CODEN: NSPUE2

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1990

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, DC 20402

PREFACE

This report constitutes the proceedings of a three day workshop on Hypertext Standardization held at the National Institute of Standards and Technology (NIST) on January 16 - 18, 1990. The workshop was the first in what we hope becomes a series of standardization efforts. The workshop was sponsored by the Hypertext Competence Project of the National Computer Systems Laboratory of NIST.

The workshop included plenary sessions and three disscussion groups. Because the participants in the workshop drew on their personal experiences, they sometimes cited specific vendors and commercial products. The inclusion or omission of a particular company or product does not imply either endorsement or criticism by NIST.

We of the Hypertext Competence Project gratefully acknowledge the assistance of all those who made the workshop a success. Further, I want to thank Dave Stotts for designing the cover graphic.

Judi Moline
January 29, 1990

PROGRAM COMMITTEE

Len Gallagher, Chairman
Jean Baronas
Dan Benigni
Richard Furuta
Judi Moline
David Stotts

CONTENTS

ABSTRACT 1

INTRODUCTION 3

REPORTS OF DISCUSSION GROUPS 5

1. HYPERTEXT MODELS DISCUSSION GROUP 7
    1.1 Reference and Data Model Group: Work Plan Status 9
    1.2 Reference and Data Model Group: Comparison of Three Models 15
    1.3 Reference and Data Model Group: Responses to 17

2. DATA INTERCHANGE DISCUSSION GROUP 19
    2.1 Summary of the Hypertext Interchange Group 21
    2.2 Note on Representing Anchors 23

3. USER REQUIREMENTS DISCUSSION GROUP 27
    3.1 Report from the User Requirements Working Group 29

PAPERS 37

1. Bornstein, J. & Riley, V. - Hypertext Interchange Format 39

2. Brown, P.J. -- Standards for Hypertext Source Files: the Experience of UNIX Guide 49

3. Cole, F. & Brown, H. -- Standards: What Can Hypertext Learn from Paper Documents? 59

4. Crane, Gregory -- Standards for a Hypermedia Database: Diachronic vs. Synchronic Concerns 71

5. Furuta, R. & Stotts, P.D. - The Trellis Hypertext Reference Model 83

6. Halasz, F. & Schwartz M. - The Dexter Hypertext Reference Model 95

7. Hardt-Komzacki, S. et al. - Standardization of Hypermedia: What's the Point? 135

8. Lange, Danny B. - A Formal Model of Hypertext 145

9. Marshall, Catherine C. - A Multi-Tiered Approach to Hypertext Integration: Negotiating Standards for a Heterogeneous Application Environment 167

10. Newcomb, Steven R. - Explanatory Cover Material for Section 7.2 of X3V1.8M/SD-7 179

11. Oren, Tim - Toward Open Hypertext: Requirements for Distributed Hypermedia Standards 189

12. Parunak, H. Van Dyke - Toward a Reference Model for Hypermedia 197

13. Riley, Victor A. - An Interchange Format for Hypertext Systems: the Intermedia Model 213

14. Thompson, Craig W. - Strawman Reference Model for Hypermedia Systems 223

APPENDICES 247

1. Kahn, Paul - Hypermedia Bibliography, 1989 249

2. Participants 265

ABSTRACT

This report constitutes the proceedings of a three day workshop on Hypertext Standardization held at the National Institute of Standards and Technology (NIST) on January 16 - 18, 1990. Efforts towards standardization of hypertext have already been initiated in various interested organizations. In recognition of these existing efforts, NIST sponsored the Hypertext Standardization Workshop organized by the Hypertext Competence Project of the National Computer Systems Laboratory.

The major purpose of the Hypertext Standardization Workshop was to provide a forum for presentation and discussion of existing and proposed approaches to hypertext standardization. The stated workshop goals were to consider hypertext system definitions, to identify viable approaches for pursuing standards, to seek commonality among alternatives whenever possible, and to make progress towards a coordinated plan for standards development, i.e. a hypertext reference model. The workshop announcement solicitated contributed papers on any aspect of hypertext standardization, including assertions that standardization is premature or inadvisable. Approximately 30 contributions were received and distributed to the 65 workshop participants on the first day.

The workshop included plenary sessions and three discussion groups. This proceedings includes the papers selected for presentation in plenary sessions, reports of the discussion groups, and supplementary materials. Major conclusions of the workshop were that the discussion groups should continue their technical efforts, and that NIST should sponsor at least one more workshop to provide a forum for public discussion of progress.

Key words: hypermedia; hypertext; standards.

INTRODUCTION

Over the past several years we have seen a significant increase in the availability of document and information management systems that call themselves Hypertext or Hypermedia implementations. These systems have received a degree of acceptance from the user community and are being integrated into an increasing number of application development projects. There is every reason to believe that this trend will continue to grow and influence the marketplace in the foreseeable future.

Although, at present, Hypertext/Hypermedia systems have no agreed formal definition, there is agreement on some of the underlying concepts that characterize them. Recently, a number of authors have stated requirements for hypertext standards and some have offered definitions and initial specifications for consideration. In several cases, specialized standardization efforts have already been initiated through interested organizations. In recognition of this emerging activity, the National Institute of Standards and Technology (NIST) sponsored the Hypertext Standardization Workshop. One consideration of the workshop was to determine if the evolution of Hypertext and Hypermedia technologies has reached the point where it makes sense to consider formal standardization.

The major purpose of the Hypertext Standardization Workshop was to provide a forum for presentation and discussion of existing and proposed approaches to hypertext standardization. We solicitated contributed papers on any aspect of hypertext standardization, including assertions that standardization is premature or inadvisable. We received approximately 30 contributions totaling more than 400 pages, which were distributed to all workshop participants on the first day. The stated workshop goals were to consider hypertext system definitions, to identify viable approaches for pursuing standards, to seek commonality among alternatives whenever possible, and to make progress towards a coordinated plan for standards development, i.e., a hypertext reference model.

Of the contributed papers, those of particularly high quality and general interest were accepted for publication and featured during a plenary session on the opening day of the workshop. Each author was given approximately 25 minutes to present a particular point of view. These individual papers are presented alphabetically in this proceedings. The remainder of the first day and all of the second day consisted of discussion groups set up in response to issues raised in the contributed papers.

Three discussion groups met in parallel on the topics of Hypertext Models, Hypertext Data Interchange, and Hypertext User Requirements. Each group chose one or more "Presentors" to convey group opinions to the whole workshop. Summaries of the deliberations and conclusions of these discussion groups, authored by the presentors, are included herein.

The morning of the third day of the workshop consisted of reports from each of the three discussion groups and a general discussion of where to go from here. In general, the groups were quite pleased with their progress and expressed a desire to meet on a somewhat regular basis to continue deliberations. There was general agreement that a recognized hypertext/hypermedia standards group could function as the focal point in defining a hypertext data model and a reference model that addresses other more specialized activities in areas such as documents, graphics, video, and sound.

Craig Thompson raised the issue of establishing a more formal hypertext/hypermedia "study group" with regular scheduled meetings and operating procedures. Possibilities for organizing such a group under the auspices of ACM, X3, IEEE, GCA, NIST, or some other ANSI accredited organization were discussed, but with no definitive conclusion. Interested individuals were encouraged to pursue possibilities within these organizations.

Major conclusions of the workshop were that the individual discussion groups should continue their respective technical efforts, possibly via private communications, and that NIST should sponsor at least one more workshop to provide a forum for public discussion of progress. A decision could then be made as to the desirability of establishing a more formal standardization group with status in some ANSI accredited standards organization.

Leonard Gallagher
Workshop Chairperson

REPORTS OF DISCUSSION GROUPS

This section of the proceedings contains the reports as submitted by the presentors of the discussion groups. The material was presented at the closing plenary of the workshop.

1. HYPERTEXT MODELS DISCUSSION GROUP

Moderator: Judi Moline

Presentors: Van Parunak
John Leggett
Jim Black

Scribe: Robert Miglin

Frank Armour
Dan Benigni
James Black
John C. Chen
Qi Fan Chen
Paul Clapis
Fred Cole
Andrew Dove
Robert Edmiston
Lawrence Fitzpatrick
Richard Furuta
Frank Halasz
Shoshana Hardt-Kornacki
Kris Houlahan
Danny Lange
John Leggett
William Loftus
Robert Miglin
Judi Moline
Howard Moncarz
Taeha Park
Van Parunak
John Puttress
Louis Roberts
Linda Rosenberg
Andrea Spinelli
David Stotts
Craig Thompson
Magda Wright

Reports of this group follow:

- Reference and Data Model Group: Work Plan Status
- Reference and Data Model Group: Comparison of Three Models
- Reference and Data Model Group: Responses to "Issues for Discussion Group Consideration"

Reference and Data Model Group (RDMG): Work Plan Status

Reported by
H. Van Dyke Parunak
Industrial Technology Institute

January 26, 1990

Abstract

A reference model is a structured description of some domain that can be used to compare existing implementations in that domain, design new implementations, and (most important for our purposes) map out possible areas for standardization and show their relation to one another. The main output of the RDMG during the NIST workshop was a work plan for arriving at such a reference model. The work plan that we propose has the following structure, where the flow of activity is down the page (except for the single feedback loop), and where activities marked by '*' received significant attention during the workshop.

```
   +-------------------+--------------------+---------+
   |                   |                    |         |
   v                   v                    v         |
*Define          *Brainstorm      *Compare Existing   |
"Hypertext"      Concepts         Models (DTL)        |
            \          |         /                    |
              \          v      /                     |
             *Organize Ontology                       |
                       |                              |
                       v                              |
           Rank Concepts by Centrality                |
                       |                              |
                       v                              |
           Inventory Existing Systems                 |
                       |                              |
                       v                              |
        Construct "Implementation" Model              |
                       |                              |
                       +------------------------------+
                       v
           Select Areas for Standards
```

The rest of this document defines each of these steps, and reports what we have done in each of them.

This document summarizes the portion of the final RDMG presentation that I delivered on 18 January 1990. It represents my perception of the deliberations of the group, but has not been reviewed or formally approved by the other members.

1 Define 'Hypertext'

This definition is intended to be a brief, succinct statement of our domain, to provide some degree of focus during subsequent stages. It may well change considerably as a result of later analysis. We began with a definition that has been circulating for several years, and modified it to reflect the valuable distinction between 'hypertext' (as a structured body of information) and 'hypertext system':

A *Hypertext* is a network of information nodes connected by means of relational links.

A *Hypertext System* is a configuration of hardware and software that presents a Hypertext to users and allows them to manage and access the information that it contains.

2 Brainstorm Concepts

In an effort to scope our discussions, we brainstormed terms and concepts describing hypermedia, and assembled a list of about 80. These are listed in more organized fashion below.

3 Compare Existing Models

In order to build on existing work, representatives of three detailed models presented at the workshop (the Dexter model, the Trellis r-model, and Danny Lange's model) compared and contrasted their respective models. A separate report by John Leggett summarizes those discussions.

4 Organize Ontology

We attempted to organize the set of terms and concepts to bring like things together. This section reviews the resulting taxonomy of concepts, and then describes some further analysis that might be conducted to organize the list even further. By itself, this organized list is a limited reference model. Subsequent steps refine it and seek to cast it in a form that has been useful in the past in guiding the development of standards.

4.1 A Preliminary Organization

We found it useful to sort the concepts produced by brainstorming into three main categories: Entities, Properties, and Functions or Operations. Some concepts did not seem to fit cleanly into any of these, and were relegated to a catch-all category, Abstractions.

Entities These are the objects that a hypertext system must manipulate; together, they make up a hypertext.

- Components, each with a UID (unique ID)
  - Link or relationship; may be warm, hot, abstract, dynamic.
  - Nodes; can have fields, contents, anchors/buttons/interactors/link markers
  - Composites, including idioms, paths, tours, webs, networks
- Whole documents, also with UID's (container, stack, frame set, guideline)
- Navigational aids, including index, map, table of contents, fisheye view
- Display entities: window, canvas. Card vs. scroll distinction applies here.
- Functional stuff: presentation specification; resolver.

Properties These can be either of entities or of the entire system.

- Properties of Entities (should probably be merged with the Entity term list)
  - Attributes (of nodes and links; includes temporal and display behavior)
  - Component format and structure (e.g., locktext)
  - Network topology (e.g., hierarchy, hypercube, DAG)
  - Size of canvas (scroll vs. card)
- Properties of the System
  - Concurrency, including both multiuser and multithread
  - Synchrony
  - Existence of a formal model
  - System performance (e.g., speed)
  - Timing (e.g., to support music, animation, and video)
  - Distributed vs. local
  - Monolithic vs. open (as in a link service or link protocol)
  - Referential integrity (are dangling links permitted?)
  - Context sensitivity
  - Interoperability
  - Operating modes (browse, author, ...)

Functions Initial attempts to classify these further were unsuccessful. We finally did a hierarchical clustering, joining the closest two items into one, and repeating until we had a reasonable number of classes. This process yielded the following taxonomy, to which we have added names that seem to summarize the contents of each group:

- Knowledge modification
  - Modifying system knowledge in place: edit (including cut/paste and structured editing), update, annotate
  - Move information into or between systems: interchange; conversion and parsing of raw text
- Navigation
  - Search and query; need for managing relevance of search; filters
  - Browsing semantics (progressive disclosure; histories; views; path macros; bookmarks)
  - Support tools: scripting, addressability, triggering (actions to take on arriving and departing a node)
- 'Yucky Systems Stuff'
  - Tailoring
  - Interfaces, of two sorts:
    - Foreign nodes (application programs that can be activated at a node); API's
    - Communications protocols (between separate programs at the same layer) and services (between layers of a single program)
  - Versioning, journaling
  - Access control

Abstractions This is a catch-all category for a number of terms that didn't seem to fit elsewhere. Alternative titles for this group of terms are 'metadata' and 'implementation tools.'

- Schema
- Type
- Class
- Object
- Data models (E-R, semantic)
- Encapsulation
- Layer

4.2 Further Organization

One can go further (though we didn't have time). For example:

- Developing a 'Properties × Functions' relation to show what functions are needed to support what (systems) properties.
- Developing an 'Entities × Functions' relation to show what entities support what functions.

5 Rank Concepts by Centrality

In choosing areas for standardization, we want to focus on those topics that are characteristic of most or all hypermedia systems, and not on those that appear only in a few systems for special purposes. The intent here is to rank each topic as $\{+, 0, -\}$ to indicate how typical or critical it is for a model of mainstream hypermedia.

6 Inventory Existing Systems

One important use of a reference model is as a guide to comparing systems, and a test of the model that this process produces will be how useful it is for such comparisons. We propose the development of a matrix showing how various existing systems reflect the categories that we have developed, as a way of testing the completeness and consistency of our ontology. Discussion in the plenary session on this point highlighted the different results that would likely be obtained depending on whether one focused on commercial systems or on research systems.

7 Construct 'Implementation' Model

The objective here is to derive a layered model, like the OSI reference model, in which the layers represent successive functionality added to a core with hardware at the bottom.

The group expressed some difference of opinion on whether OSI is a good example of what we want.

An interesting discussion within the group centered on whether a monotonic layering from hardware to application was possible. One suggestion was that in fact there might be several implementation stacks, doing different tasks, for instance:

```
                              S T A C K S

TASK:      Store         | Process      | Present         | User
                         |              |                 |
LAYERS:    Node,Link     | Navigate     | Window,Button   | Concept
                         |              |                 |
           - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                         |              |                 |
           OODB          |              | Virtual         |
                         |              | Terminal        |
           File System   |              |                 |
                         |              |                 |
DEVICE:   Disk           | CPU          | CRT/Keyboard    | Eye/
                         |              |                 | Hand
                         |              |                 |
                       \   /          \   /       \               /
MEDIUM:                 Bus            LAN         EM Radiation
```

The layers listed in this diagram are incomplete, but illustrate the difference between those that are central to hypermedia and must be described in our model (above the dashed line), and those that should be developed in other disciplines (below the line). What is critical for our purposes is the clear definition of the services that connect one layer to another.

8 Select Areas for Standards

Once developed, a reference model helps map out areas for standards. Focus is important here, and the model helps provide it in two dimensions. The ranking of concepts in the ontology shows how central each is to hypermedia, and helps us focus on standardizing those concepts most likely to be of widespread use. The implementation model helps us identify which concepts are best standardized in other research communities (such as CHI, DB, OOPS, windowing systems) and which require the focused attention of researchers in hypertext. Graphically, the focussing process seeks to identify the region 'X' in the diagram below for standardization.

```
                     |   |
                CHI  |   |
                     + - V - - - - - - - - - - - - - - - - - -
Which           HT   |   X  <---
Community?           + - ^ - - - - - - - - - - - - - - - - - -
                DB,  |   |
                OOPS |   |
                     +--------------------------------------
                     All HT                           Few HT
                     Systems                          Systems
                         How central is it to hypertext?
```

Reference and Data Model Group: Comparison of Three Models

John J. Leggett
Department of Computer Science
Texas A&M University

The Reference and Data Model working group spent 45 minutes comparing and contrasting the R-model[1], Dexter[2] and Lange[3] reference models. David Stotts, Danny Lange and John Leggett spent another 90 minutes over dinner discussing the three models. A summary was provided by John Leggett during the final plenary session. As these three models are currently under development, the comparisons are rather broad in nature. It is interesting to note that the three models were developed independently and with varying levels of collaboration. The results of these discussions are presented below in mostly tabular form.

Differences

	Type	Links	Anchors	Formalized?
R-model	Meta-model for systems specification	No links, but relations defined	No distinct anchors	No
Lange	Model of hypertext	Allows dangling links	Anchors and regions	Yes, in VDM
Dexter	Model of hypertext systems	Does not allow dangling links	Anchors	Yes, in Z

Similarities

Support for ***types*** in all three models is through arbitrary attribute/value pairs.
All three models have separated ***content***, ***structure*** and ***presentation***:

	Content	Structure	Presentation
R-model	Abstract content	Structure and abstract containers	Concrete and visible levels
Lange	Schema	Networks and structures	Unspecified
Dexter	Within-component layer	Storage layer	Run-time layer with presentation specifications

---

[1] Richard Furuta and P. David Stotts, "The Trellis Hypertext Reference Model," these proceedings.

[2] Frank Halasz and Mayer Schwartz, "The Dexter Hypertext Reference Model," these proceedings.

[3] Danny B. Lange, "A Formal Model of Hypertext," these proceedings.

Hypertext Reference Model Group
Responses to "Issues for Discussion Group Consideration"

James Black

1. What is the current state-of-affairs in this topic area? What is likely to happen in the near future?

The Reference Model Working Group did a reasonably thorough examination of three independently derived hypertext models and identified no essential inconsistencies which would preclude eventual consensus. Each of the three models was the product of a different analytical approach and there remain significant areas of confusion and lack of current consensus which seem to largely due to syntactical differences. Further open dialogue among the participants would improve this situation.

2. Are emerging technologies driving this topic in a certain direction? Is there sufficient stability to warrant further pursuit of standardization at this time?

The sessions revealed no clear evidence that "emerging technology" was driving any aspect of the hypertext concept in a particular direction. The only indication of any "driving forces" which may be prematurely affecting aspects of the evolution of hypertext technology are related to other standardization efforts, specifically, ODA and 5G@. There does seem to be sufficient stability in the shared understanding of basic hypertext concepts to warrant further pursuit of standardization.

3. What are the most important concepts? Ale there agreed definitions? Is there a glossary available, or set of candidate key words?

The essential concepts of hypertext would include a data model with the following features:

- data type and media independence
- "format" and "content" independence
- freely defined, relational links between freely defined data elements
- no inherently hierarchical structure
- distinct separation of format and content

They would also include such functional features as navigational, authoring, presentation, and systems management tools.

4. What is the interdependency of this topic area with other topic areas identified at this workshop?

There is a need to develop a glossary and taxonomy of hypertext terminology which includes formal, (mathematical) definitions where available. There is available a core set of candidate key words.

5. What are the major problems and controversies? Is compromise possible? or would alternative approaches better serve the vendor/user communities?

There is significant interdependency between the hypertext reference model and system interchange issues.

6. What is the ultimate goal for this topic area? a user guideline? a domestic standard? an international standard? something else? What is an appropriate sequence of steps leading to this goal?

The ultimate goal of this working group is to establish a hypertext system reference model and use it to establish a hypertext glossary and taxonomy and to identify candidate areas for standardization activity.

7. What concepts in this area are appropriate for standardization? What concepts are not appropriate for standards? What can inhibit the development of standards? Is something ready for standardization at this time?

There are no areas ready for standardization at this time.

8. What role can NIST play in achieving the goals of this topic? Are further workshops desirable? What is the most appropriate follow-on activity after this workshop?

NIST can establish a formal, on-going hypertext study group that publishes consensus findings and recommendations which NIST links to relevant standards organizations.

2. DATA INTERCHANGE DISCUSSION GROUP

Moderator:	Len Gallagher
Presentor:	Tim Oren
Scribe:	Jan Walker

Rob Akscyn
Gregory Crane
Valerie Florance
Edward A. Fox
David Fristrom
Len Gallagher
Steve Newcomb
Charles Nicholas
Tim Oren
Kenneth Pugh
Victor Riley
Jan Walker

Reports of this group follow:

- Summary of the Hypertext Interchange Group
- Note on Representing Anchors

Summary of the Hypertext Interchange Group

The Interchange Group first discussed how the problem could be partitioned. We agreed that ideally the representation of the data and its presentation to the user should be separated. However, for efficiency reasons most existing hypertext systems which support graphics in fact store bit maps and specific screen coordinates. This is an obstacle to interchange between platforms with differing display architectures.

We also made the distinction between a "delivery interchange" standard and an "archival interchange" standard. A delivery interchange standard would be directly usable by a conforming hypertext system without translation. We regarded this as very difficult to achieve in the short term due to differences in hypertext systems' methods of storing and indexing their data, which are usually highly optimized for the particular platform and application. The dependence on display formats already noted is also an obstacle to a delivery interchange standard.

An "archival interchange" standard is one in which the information owner may store hypertext in a system independent fashion. For actual delivery either the information owner or end user would need to translate the archival interchange format into a format specific to a particular hypertext software/hardware configuration. Any changes authored by the end user would have to be rolled back to the archival store before reaching other platforms, rather than attempting direct interchange. We agreed that this goal was more achievable in the short run, and turned our discussion in this direction, but without disputing the eventual value of a delivery interchange format, or the need for further experiments with delivery to define requirements for the archival representation.

We proceeded to compare relevant interchange proposals from the working papers or which were otherwise drawn to the attention of the group. These included a discussion paper submitted by Ken Pugh, Victor Riley's Intermedia exchange paper, portions of the HyTime proposal, and the so-called "HIP" Hypertext Interchange Protocol developed at Apple, Xerox PARC and Brown IRIS. A copy of the HIP paper was supplied by Victor Riley of Brown IRIS. The group voted to request that the HIP paper (Bornstein and Riley, "Hypertext Interchange Format") and relevant sections of HyTime (Newcomb, "Explanatory Cover..." and Section 7.2) be included in the final Proceedings of the NIST Workshop.

Comparing these formats showed that all were adopting a partitioning of the problem into data objects, anchors, and links. Anchors form the data object type specific endpoints for links. While there were abundant differences in terminology, a first reading showed basic conformance to this layering, and we agreed that this should be drawn to the attention of the modeling group.

It was also noted that most of the interchange proposals used SGML or SGML-like markups. After some discussion, it was agreed that SGML was a reasonable basis for further interchange experiments. This position is adopted without prejudice to an eventual standard, due to a number of participants' concerns about technical issues (e.g., efficiency, limits of a parser driven implementation), and prejudgment of the decision process. We agreed that documents resulting from these discussions should be conveyed to the HyTime (ANSI X3V1.M8) committee for inclusion in their working document set.

A general discussion of related standards ensued. There was consensus that wherever possible hypertext interchange standards should incorporate existing media type standards without requiring changes in those standards.

An ad hoc group composed of Ed Fox, Steve Newcomb, Tim Oren, and Victor Riley met during the evening to continue the comparison of the various interchange proposals. They reported to the whole group that they had succeeded in a first pass reconciliation of the anchor levels of HIP, Intermedia and HyTime. Their notes are appended in the interchange section of the proceedings (under the title "Note on Representing Anchors") rather than incorporated here, as they were not a result of the entire group.

The whole group strongly suggests that further experiments with interchange between existing systems be undertaken. We noted the need for a publicly available, editorially controlled document set for this purpose. This should be in the few hundred to few thousand node size, marked up in SGML with linking information provided. Further volunteers and funding for these experiments are an issue. Availability of a free or inexpensive SGML parser is required if universities are to participate in the experiments.

We identified a number of significant issues which were not addressed due to time constraints:

- Making a complete list of relevant data type standards
- Requirement for unique naming and identification services, which is a problem with wider scope than hypertext alone.
- Link typing, type definition and hierarchies, N-way link structures
- Composites - a taxonomy of existing uses and representations
- Versioning
- Representation of time-based media, e.g., music, video, and links conveying timing information

These should be addressed in further sessions, as they all influence requirements for an interchange standard and some (particularly link typing and composites) are the subject of active research and controversy.

Submitted by Tim Oren
January 24, 1990

Note on Representing Anchors

Reported by Tim Oren

An ad hoc subgroup of the Interchange working group met to compare various proposals for archival interchange. It was composed of Ed Fox, Steve Newcomb, Tim Oren, and Victor Riley. These notes are the result of that meeting. They are a first pass which has not been considered by any other group. See the summary of the Interchange group for context and definition of terms.

We chose to proceed by focusing on the anchor or "anchor-like" portion of each proposal. We began by considering how the features of the Intermedia Interchange could be added to the HIP proposal, and expressed the result in HIP-like terms. We then attempted to reconcile this result with the formalism and language of the pertinent sections of HyTime. Note that this applies only to anchors, and there may be additional difficulties in reconciling layering strategies when we look at the link layers of the various proposals.

1. Reconciliation of Intermedia exchange and HIP

This is a semi-formal presentation of patches to the <ANCHOR> section of the HIP specification. The other sections of HIP have not yet been brought into conformance:

<NAME> - optional, ASCII string, user displayed or for use of system. Usage ideas: this could be the name of a hypercard button, or used as a item for searching, or as comments to be displayed as preview.

<ID> - required, a unique ID in a format TBD. Uniquely identifies this anchor within the scope of the interchange set.

<CREATION> - optional.

> <WHEN> - Date/time of creation in a standard form TBD. Indicates the moment of original creation of the anchor (even if it was later moved).
>
> <BY> - the unique ID (TBD) of the user/authority who created the anchor.

<MODIFIED>* - optional, optionally multiple.

> <WHEN> - Date/time of the particular modify. It is a application policy matter whether all, just the latest, or no mods are recorded.

<BY> - the unique id of the modifying user/authority.

<VERSION> - a unique id of the referenced version. How to use this is a policy matter of the system. If it's the same as the <ID> of this anchor, this is the current version.

<LOCATION> - required.

<ANCHOR-OBJECT-ID> - required. The unique ID of the data object (file - chunk - whatever) to which this anchor refers.

<ANCHOR-VALUE>+ - object type specific, required, optionally multiple. Note that this could refer to multiple selections, elements, etc. within the data object.

<PRESENT-SPEC> - object type specific, optional, regulates how the anchor is to be presented, e.g., run the sound editor or play the sound, positioning information for the 3-D editor view of an IGES object.

2. Reconciliation with HyTime terminology (sections under 7.2.5)

HyTime as written contains within its "location" layer information which is both generic to the concept of anchor, and specific to certain data types. We try to separate this here. Again, this has not been reconciled with the link layer of HyTime or HIP and problems might emerge there.

The general concept of "entloc" corresponds to the HIP <ANCHOR> idea. The ID within entloc corresponds directly to the <ID> in HIP. The "dataent" corresponds to the <ANCHOR-OBJECT-ID> of HIP.

Notation Data Location (ndloc) is HyTime's generic anchor, corresponding directly to the HIP construct above. Its type specific part is represented in the "formula," which corresponds to the <ANCHOR-VALUE> of HIP. "Snap" should probably be considered part of a type-specific construct rather than part of a generic anchor. HIP would probably represent it as part of the <PRESENT-SPEC>. A reasonable default data type is undifferentiated byte stream.

The other location constructs are viewed as data type specific anchors.

Character data set location (cdloc) is an anchor into sequences of ISO defined characters (NB: this is not the same thing as a font or byte sequence).

Document locations (elemloc) (7.2.5.2-3) are the SGML object type specific anchor definitions. Element location is SGML type specific and identifies a single "node" within the hierarchical structure created by an SGML markup. This may be specified using an ID, if one exists for the node, or using a path designator from the root. Point location allows anchoring to a spot within an element.

All of these constructs might be further generalized by allowing multiple "selections" to be incorporated within one "location."

3. USER REQUIREMENTS DISCUSSION GROUP

Moderator:	Jean Baronas
Presentor:	Robert Glushko
Scribe:	Seymour Hanfling

Carol Adams
Peter Aiken
Jean Baronas
Denise Bedgord
Tim Berners-Lee
Kevin Gamble
Robert Glushko
Louis Gomez
Seymour Hanfling
Casey Malcolm
Catherine Marshall
Fontaine Moore
Dan Olson
Duane Stone
Clifford Urr
David Wojick
Don Young

Reports of this group follow:

- Report from the User Requirements Working Group

REPORT FROM THE USER REQUIREMENTS WORKING GROUP

Robert J. Glushko
Search Technology
Norcross, GA

This report summarizes meetings held on January 16-17, 1990 during a workshop on Hypermedia Standardization held at the National Institute of Standards and Technology in Gaithersburg, MD. In addition to the author, the members of the Working Group for User Requirements were Carol Adams, Peter Aiken, Jean Baronas, Denise Bedford, Tim Berners-Lee, Valerie Florence, Kevin Gamble, Louis Gomez, Seymour Hanfling, Kathryn Malcolm, Cathy Marshall, Fontaine Moore, Dan Olson, Duane Stone, Clifford Uhr, David Wojick and Don Young. The group followed an agenda set by NIST to identify the current state of affairs, important driving and constraining factors, potential areas for standardization, and research needs.

Complete consensus on these complex topics was impossible in two days for a group this size, so this report emphasizes the majority themes for the issues that received the most attention. I apologize for my own biases, which undoubtedly show through.

THE CURRENT STATE OF AFFAIRS FOR HYPERTEXT

In recent years hypertext concepts for making information more accessible and usable have been applied to a bewildering variety of applications:

Reference books, encyclopedias, dictionaries

Library collections and archival literature

Online software reference manuals

Policies, procedures, regulations

Maintenance and diagnostic information

Online help systems and embedded training

Education, tutorials

Engineering and CAD

Professional project management

Collaborative problem-solving and authoring

Interactive fiction, entertainment

Museum directories and information kiosks.

Four basic factors appear to account for the rapid spread of hypertext design concepts. These are enabling technology, documentation standards initiatives with hypertext implications, market pressure, and academic interest.

Enabling technology. Hypertext applications require a significant amount of local processing power and storage capacity that until the mid 1980s was not readily available. Hypertext (and especially hypermedia) applications are also benefiting from increased data transfer capabilities enabled by advances in data compression, fiber optics, and progress toward an end-to-end digital telecommunications network. Nevertheless, having the delivery and storage technology base for hypertext systems would have been meaningless without the concurrent maturation of user interface design concepts and tools. Object-oriented programming and prototyping toolkits that embody direct manipulation user interface concepts make it possible to design and implement the rich functionality of hypertext systems in a cost-effective way.

Documentation standards initiatives with hypertext implications. Some major standards efforts in related areas have made hypertext both more necessary and more likely. The first of these is SGML, the Standard Generalized Markup Language [7]. In 1986 SGML became an international standard (ISO 8879) for defining the logical structure of printed documents independently of their appearance. While there is no agreement that SGML is the optimal starting point for a hypertext standard, there is little dispute that SGML's system-independent markup makes it significantly easier to exchange and process electronic documents and hence, to combine them into hypertext documents.

A second major standards initiative that is emerging as a driving force for hypertext is CALS, the U.S. Department of Defense program for Computer-Aided Acquisition and Logistic Support [3]. CALS has as its goal the creation of a "paperless environment" with the integration of the various "islands of automation" that participate in the system design, development, deployment, and maintenance processes. In February 1988 the CALS program adopted SGML as a military standard (MIL-M-28001) for the digital form of traditional printed documents, but new standards for creating, exchanging, and delivering information are evolving that completely do away with any notion of "printed page." Since so many companies do business either directly or indirectly with the Department of Defense, the scope of CALS will be enormous. The obvious benefits of digital information exchange throughout the entire government are causing CALS concepts and requirements to spill over into other parts of government.

Market pressure. Programs that called attention to their hypertext features had started to emerge in the mid-1980s, but since the release and aggressive marketing of HyperCard by Apple Computer in 1987, dozens of other software products that claim to provide hypertext and hypermedia capabilities have entered the marketplace since.

Academic interest. Finally, substantial academic interest in hypertext issues has emerged in the last few years. In late 1987, approximately the same time as the introduction of HyperCard, a conference was held at the University of North Carolina that was the first academic rally of researchers and system designers under the hypertext flag [1]. Since then, similar conferences have been held in Europe [9] and a second major conference on hypertext was held in Pittsburgh in November 1989 [2]. At least one new professional journal has been established with "hyper" in its name [6].

THE FUTURE

The 1990s will see ubiquitous software and hardware support for hypermedia applications in "off the shelf" computing environments. Computer hardware, software, and telecommunications companies will develop business strategies and product lines for multimedia systems, applications, and services.

It is already readily apparent that no single hypertext design or hypertext software is appropriate for all applications or users. However, guidelines or standards for choosing design approaches or software tools are hard to apply without a framework for understanding the range of possible applications into which hypertext solutions might fit.

NEW VIEWS OF THE HYPERTEXT "DESIGN SPACE"

Nevertheless, the classification scheme for hypertext applications that this paper began with is too arbitrary to serve this important purpose. That scheme loosely categorizes hypertext applications according to the kind of information they contain, but has no rationale for defining the categories. Why aren't encyclopedias and dictionaries in their own categories? Shouldn't training and education be together? Clearly, a more abstract and robust scheme is needed for comparing, understanding, and generating hypertext applications. The working group discussed several alternative views of the "hypertext design space."

Dimensional view

An alternative that I have been developing is based on four non-orthogonal dimensions:

User dimension: single users vs. groups vs. multiple unrelated users. Hypertext systems can be designed for single users, groups of users working collaboratively, or large communities of unrelated users.

Information dimension: creation vs. conversion. Hypertexts can primarily contain new information created for the application or information obtained by converting information that already exists in conventional printed form.

Task dimension: task-specific vs. general. Hypertext systems can be designed to support specific tasks or as general-purpose environments for building other hypertexts.

Interface dimension: static vs. dynamic. Hypertexts can be primarily static archives for read-only browsing, can be relatively transient databases of periodically-published information like news articles or product catalogs, or dynamic to support continuous collaborative authoring and commentary.

To edit, or not to edit?

An alternative framework for understanding the hypertext design space was proposed by Carol Adams. Her view is that all hypertext applications can be partitioned according to whether or not they allow users to edit the content of the basic hypertext units and the links between them. These two orthogonal dimensions yield four cells into which existing and potential hypertext applications might be categorized.

The two clearest categories in this framework are applications in which both units and links can be edited, and "read-only" or pure "browsing" applications in which neither can. Applications of hypertext to software design or concurrent engineering domains might embody a fixed structure between unit templates and thus primarily support unit-only editing. Finally, applications that involve primarily link-only editing with permanent units might include archives or literary criticism.

SPECIFICATION OF HYPERTEXT FUNCTIONS

Standards for the appearance of hypertext user interfaces may not even be possible and are certainly premature. The range of applications that call themselves hypertext and the wide assortment of user interfaces they contain clearly argue that at best, subsets of standards or standards "families" would be appropriate. However, the working group concluded that users and application developers would benefit immediately from shared definitions and specifications for hypertext functions. "Functions" are defined here as operations carried out by a hypertext user interface on the entities managed by the hypertext storage layer [5].

The goals of specifications for hypertext functions are straightforward. They must:

a) fit clearly into the hypertext reference model,

b) be independent of presentation specifications, and

c) unambiguously define the operational semantics.

If these goals can be satisfied, perhaps standards for hypertext functions can emerge that can be organized into consistent subsets for different parts of the hypertext design space. Then, the interoperability of hypertext systems in the same region of the design space can be defined in terms of these functions. The working group began this ambitious effort by creating a list of functions and crudely separating them into "authoring" and "reader" subsets. No claim is made that these lists are complete.

Authoring Functions

1) Create (unit, link, composite)

2) Edit (unit, link, composite)
3) Delete (unit, link, composite)
4) Publish (unit, link, composite, hypertext). "Publish" means to give a hypertext component a degree of permanence in some current version or configuration of the storage layer.

Reader Functions

1) Indicate current unit
2) Move to another unit
    a) defined spatially (e.g., arbitrary new location in display)
    b) defined syntactically (e.g., in order -- "next," "back")
    c) defined lexically (e.g., unit name contains string "x")
    d) defined semantically (e.g., unit of type "x")
    e) defined temporally (e.g., previous current unit)
3) Indicate presence of "expandable" structure
4) Indicate whether currently expanded
5) Expand current unit
6) Close current unit

Annotation Functions

7) Create annotation
8) Edit annotation
9) Delete annotation

Bookmark Functions

10) Create bookmark
    a) implicitly when in unit
    b) explicitly by user action
11) Delete bookmark
12) Move to "book-marked unit"

Functions on Virtual Structures

13) Search (scope, specification)
14) Define session (history, bookmarks, annotations)

15) Save session
16) Restore session

Miscellaneous Functions

17) Print (Unit, link, linearization)

Specifying Functional Semantics

These lists of functions will be far more useful when accompanied by precise definitions of what they mean and the rules by which they can be combined. There are many notations for specifying the semantics of functions (e.g., [4]), but I will use an informal approach here that is commensurate with the rudimentary level of the working group's progress in developing the specifications.

For example, BACK (NEXT (X)) = X defines the meaning of "NEXT" and "BACK" functions in a hypertext system as follows: if a reader navigates from a unit X using a "NEXT" function, the "BACK" function returns to the starting unit X.

Similarly, DELETE (CREATE (X)) = CREATE (DELETE (X)).

But, DELETE (PUBLISH (CREATE (X))) is not equal to DELETE (CREATE (X)), because the intervening "PUBLISH" function defines a different version or configuration of the hypertext.

RESEARCH AGENDA

The working group concluded that research is needed in many cases to help define the appropriate semantics for hypertext functions, and it would be appropriate for NIST to conduct, sponsor, or encourage this research. Research is also needed to define new measures for hypertext that describe characteristics relevant to user performance. This research agenda should include research into these areas:

Evaluating "hypertextability." While there are informal guidelines for determining whether a particular document or document collection is suitable for conversion to hypertext, more reliable and objective measures are needed. "Hypertextability" can potentially be characterized by aspects of the logical structure of a document, such as the number, size, and relationships of the information units.

Validation of hypertext conversion. Measures of hypertextability will also be invaluable in hypertext projects for estimating the resources required and estimating schedules. Corresponding methods and tools for measuring the "amount of hypertext" that has been successfully converted should follow; perhaps hypertext sets of links can be evaluated using analogues to the familiar ideas of "precision" and "recall" in information retrieval.

Measuring hypertext "readability." Readability formulas for ordinary text based on sentence length, word length, or other characteristics have been a continuing subject of research [8]. Hypertext extensions to readability metrics might include measures of the "goodness" of links based on similarity between linked units. Readability measures for alternative hypertext designs for the same text will go far toward making hypertext design an engineering discipline.

A final research area identified by the working group where progress will immediately benefit users involves intellectual property issues for hypertext and hypermedia. The rash of "look and feel" copyright infringement lawsuits and similar claims for software patents confront software designers and developers with chaos, uncertainty, and legal action [10]. But as unclear as the situation is for software in general, the novel character of hypertext and hypermedia software raises still more complexities for intellectual property law. For example, if copyright law has different rules for "literary works," "audiovisual works," "sound recordings," and "pictorial works," into what legal category does an interactive hypermedia encyclopedia or a talking book fall? Are new links or notes in a hypertext system considered "derivative works" under copyright law? These and other issues are not just legal curiosities -- they will have considerable impact on the legal protection available and hence the economic viability of hypermedia systems.

REFERENCES

[1] Association for Computing Machinery. *Hypertext '87 Proceedings.* ACM: New York, 1987.

[2] Association for Computing Machinery. *Hypertext '89 Proceedings.* ACM: New York, 1989.

[3] Department of Defense. *Computer-aided Acquisition and Logistic Support.* Office of the Secretary of Defense CALS Office, The Pentagon, Room 2B322, Washington, D.C. 20301.

[4] Guttag, J. Abstract data types and the development of data structures. Communications of the ACM, 20(6), June 1977.

[5] Halasz, F., and Schwartz, M. The Dexter hypertext reference model. *Proceedings of the NIST Hypertext Standardization Workshop*, Gaithersburg, MD, January 16-18, 1990.

[6] *Hypermedia.* 1(1), 1989.

[7] International Organization for Standardization. *Standard Generalized Markup Language,* ISO 8879-1986.

[8] Klare, G. Assessing readability. *Reading Research Quarterly*, 1974-1975, 10, 62-102.

[9] McAleese, R. (Ed.). *Hypertext: Theory into practice.* Blackwell Scientific, 1989.

[10] Samuelson, P. Protecting user interfaces through copyright: The debate. *Proceedings of the ACM Conference on Computer-Human Interaction - CHI '89*, 97-103.

PAPERS

This section of the proceedings contains the twelve contributed papers which were accepted for publication and featured during the plenary session on the opening day of the workshop. It also contains the two papers which the interchange group recommended be added.

Hypertext Interchange Format
-- Discussion and Format Specification --
DRAFT 1.3.4

jeremy bornstein
victor riley

The Hypertext interchange format described here is based on the work of the Dexter group, an industry coalition of hypertext researchers interested in a standard for hypertext data exchange. This paper describes the result of a collaboration towards this end between Jeremy Bornstein and Frank Halasz, with significant input from other members of the Dexter group, most notably Tim Oren. The work took place during the summer of 1989, and a demonstration is planned for the Hypertext '89 conference in November of 1989.

background and rationale

The number of hypertext platforms is increasing, not decreasing. Although this development will most likely settle down to a stable state, it is almost certain that no one platform will dominate the hypertext world to the extent that nobody at all will use an incompatible platform. Nevertheless, large bodies of hypertext data are being developed in systems which will either die or evolve. An interchange format allows users on separate systems to share their data, thus eliminating the need to acquire, learn, and use a new hypertext system only to access that system's data.

Of course, in order to propose a reasonable interchange format, the structure of the data must first be determined. As it happens, with regard to hypertext this is by no means a closed issue. The Dexter group made the decision to describe a format which would be able to include everyone's definition of hypertext and thereby short-circuit "rathole" debates about the nature of hypertext, instead focusing effort on the structure of a given system's hypertext. The framework, described below, attempts to be an inclusive definition rather than an exclusive one.

generalities

The format is an ASCII format, as opposed to a binary format. Conversion to a binary format is possible if desired, but a text format is much easier when the definition of the format is still evolving.

The appearance of the format is similar to that of SGML[1]: there are tags marking the beginning of a hierarchical section and tags marking the end ("begin-tags" and "end-tags"); the end-tag corresponding to a given begin-tag has a backslash ("\") in front of the name for the begin-tag. Tags appear between greater-than and less-than signs ("<" and ">"); if the greater-than sign appears in the data, it is doubled ("<<"). The order of the children of a given tag is irrelevant[2].

Tags which are not understood by a parser are guaranteed to be ignored by that parser. In other words, if a particular system exports information which no other system understands (yet), then this will not cause another parser to crash, but merely render an incomplete version of the document.

The characters A-Z, a-z, 1-9, and the underscore ("_") are the only valid characters which may be used in the name of a tag. Case is not significant. So far, the agreed-upon conventions are that tags begin with a lower case letter and that words after the first are marked by capitalization of the initial letter. For example, "thisHasFourWords" is a tag name which adheres to these conventions.

Whitespace, when it appears outside of the data belonging to a bottom-level tag, is not significant. Often in examples, a single space character is added after bottom level start-tags and before the corresponding end-tags, but this whitespace is not in the actual export files. The indentation which appears in examples is also not part of the format, but it should not cause an interchange-format parser to fail.

Since many references in a hypertext environment will take place across "document" boundaries, it is necessary to be able to reference many objects from a global standpoint. In order to make this independent of file name and directory position, global IDs are used. So far, the numbers are 64 bit numbers which may be chosen by any method, preferably including at least some random bits. Eventually this may be changed in favor of some method which better ensures uniqueness of each identifier.

specifics

---

[1] SGML -- Standard Generalized Markup Language

[2] That is, the following two expressions are equivalent:

- <foo> <bar> 128 <\bar>
  <baz> 256 <\baz> <\foo>
- <foo> <baz> 256 <\baz>
  <bar> 128 <\bar> <\foo>

This section is a rather humorless and redundant description of the data format. It might be more efficient to read the sample file first and then refer below for confirmation and clarification of your understanding. The description which follows is hierarchical, as is the interchange format itself.

<DOCUMENT>

The outermost tag in a HIP-format document is the <DOCUMENT> tag. The <DOCUMENT> tag has four possible types of children: the <HEADER> tag, <NODE> tags, <LINK> tags, and <COMPOSITE> tags.

<HEADER>

The <HEADER> tag contains relevant information about the document as a document: the name, the unique id, which system it was exported from and on what date.

<NAME>

This is the name of the document in the originating system. The name is primarily for display to the user, but it is possible that it could be used in trying to resolve links as well.

<ID>

This is the unique id of the document, following the rules for ids given above.

<EXPORTED>

This tag contains information about the originating system and when the document was exported from that system.

<FROM>

This is the name of the originating system.

<DATE>

This is the date on which the document was exported. A standard format for the date has not been agreed upon.

<ACCESS>

These are the access rights for the document set. In the case of Intermedia this is the web, for NoteCards this is the NoteFile, for HyperCard this is the stack. No format has been agreed upon.

<CREATION>

The <CREATION> tag tells the time of creation and the creator for the document.

<BY>

This is the creation author.

<DATE>

This is the date which the document was created.

<MODIFIED>

The <MODIFIED> tag tells the time of modification and the modifier for the document. A set of these can tell history for changes.

<BY>

This is the modifier author.

<DATE>

This is the date which the document was last modified.

<NODE>

The <NODE> tags in a document function as the wrappers for the text/graphics/&c. A <NODE> has several parts:

<USE>

This tag is used to specify the location to the contents of the NODE. If two <DOCUMENTS> share the same <NODE>, the <USE> tag is used to specify the location of the shared data.

<NAME>

This is the name of the node in the originating system. The name is primarily for display to the user.

<ID>

This is the unique id of the <NODE> (see above).

<ACCESS>

These are the access rights for the node.

<CREATION>

The <CREATION> tag tells the time of creation and the creator for the node.

<BY>

This is the creation author.

<DATE>

This is the date which the node was created.

<MODIFIED>

The <MODIFIED> tag contains information about who made the last modification to the NODE, and when the modification was made. A set of these can tell history for changes.

<BY>

This is the userid (or other identifying information) of the last person to modify the NODE.

<DATE>

This is the date which the node was last modified.

<DATA>

The <DATA> tag contains the <NODE>'s low-level data (text or a picture, for example). If the <USE> tag is used, this should be NULL.

<runTimeStuff>

The <runTimeStuff> tag contains information about how the <DATA> should be displayed; it is currently the tag undergoing the most revision. It is expected that much of the information within it, such as font name, will often be unusable in the imported-to system. Within the <RunTimeStuff> tag, the five tags below are the only ones currently defined. The last three will most likely be uninterpreted by any system besides HyperCard.

<FRAME>

The position of a NODE with respect to its parent[3] is described by the <FRAME> tag. If the <FRAME> tag is absent, then the parent <NODE> is considered to be "immediately subsequent" to the previous <NODE>. This would be the case for multiple <NODE>s in a creamy hypertext system such as Notecards or InterMedia. Otherwise, the following two tags determine the frame:

<SIZE>

This is the size (x,y) of the node.

<LOCATION>

This represents the offset (x,y) between the parent's origin and the node's origin. If not present, it is undefined and the importing system is free to set it arbitrarily.

---

[3] The parent may be a <COMPOSITE> node or null.

<fontSpec>

The <fontSpec> contains information about the font of the data.

<NAME>

This tag contains the name of the font.

<SIZE>

This tag contains the point size of the font.

<STYLE>

This tag contains any style modifications to the font: i.e., bold, italic, underline, &c.

<JUSTIFY>

This tag contains the justification rule for the text: left, center, or right.

<lockText>

This tag is "true" if the user is allowed to modify the text of the item, and "false" otherwise.

<STYLE>

This tag, probably only interpreted by HyperCard, describes the frame for the <NODE>'s <DATA>.

<originalType>

This tag, also probably only interpreted by HyperCard, contains "button" or "field," depending on the original type of the object.

<ANCHOR>

There may be several <ANCHOR> tags within a given <NODE>. The anchor tags contain information about all anchors present within the <NODE>'s <DATA>.

<NAME>

This is the name of the anchor in the originating system. The name is primarily for display to the user.

<ID>

This is the unique id of the anchor and must be present.

<CREATION>

The <CREATION> tag tells the time of creation and the creator for the anchor.

<BY>

This is the creation author.

<DATE>

This is the date which the anchor was created.

<MODIFIED>

The <MODIFIED> tag contains information about who made the last modification to the ANCHOR, and when the modification was made. A set of these can tell history for changes.

<BY>

This is the userid (or other identifying information) of the last person to modify the ANCHOR.

<DATE>

This is the date which the anchor was last modified.

<LOCATION>

This is the offset in bytes (O is the position before the first character) of the anchor text. If the <LOCATION> is a pair of numbers separated by a comma (or a triple for 3-D space), this describes the text span already in the <DATA>. If the <LOCATION> is absent, the whole <DATA> is the relevant text.

<TEXT>

This is the text which the anchor is attached to. If the <LOCATION> tag is a single number (i.e., no comma) then the text is inserted at that position. Otherwise, the text need not be specified.

<runTimeStuff>

The <runTimeStuff> tag contains information about how the <ANCHOR> should be displayed; it is currently undergoing revision.

<VIEW>

The <VIEW> tag contains information about how the <ANCHOR> could be viewed. This also specifies whether the <ANCHOR> is a 2D or 3D view or either. Right now, this is application specific.

<OBJECT>

The <OBJECT> tag specifies the objects the <ANCHOR> is attached to. This covers multiple spans of text, or multiple graphical objects. Right now this is application specific.

<LINK>

A <LINK> holds all the information about a single bidirectional link. This may be expanded in the future to describe multi-headed and multi-tailed links.

- **<NAME>**
  - This is the name of the link in the originating system. The name is primarily for display to the user.
- **<ID>**
  - This is the unique ID of the link itself.
- **<sourceNodeId>**
  - This is the ID of the node associated with the start of the link.
- **<sourceAnchorId>**
  - This is the ID of the anchor (within the source NODE) from which the link originates. If unspecified, the link is from the whole NODE.
- **<destinationNodeId>**
  - This is the ID of the node associated with the end of the link.
- **<destinationAnchorId>**
  - This is the ID of the anchor (within the destination NODE) to which the link is bound. If unspecified, the link destination is the whole NODE.
- **<CREATION>**
  - The <CREATION> tag tells the time of creation and the creator for the link.
  - **<BY>**
    - This is the creation author.
  - **<DATE>**
    - This is the date which the link was created.
- **<MODIFIED>**
  - The <MODIFIED> tag contains information about who made the last modification to the LINK, and when the modification was made. A set of these can tell history for changes.
  - **<BY>**
    - This is the userid (or other identifying information) of the last person to modify the LINK.
  - **<DATE>**
    - This is the date which the link was last modified.
- **<TYPE>**
  - This is a string which describes the type of link; some examples: "Explanation," "Next," "Annotation."

<COMPOSITE>

A <COMPOSITE> tag is the framework within which frame-based systems such as HyperCard and KMS represent cards/frames. It contains an <id>, one or more <NODE>s, and a <runTimeStuff>.

- **<ID>**
  - This is the <COMPOSITE>'s unique ID.

<runTimeStuff>

So far, the only <runTimeStuff> defined for a <COMPOSITE> is the <FRAME>.

<FRAME>

The <FRAME> represents the <COMPOSITES>'s size and relation to its parent.

<SIZE>

This is the size (x,y) of the composite.

<LOCATION>

This represents the offset (x,y) between the parent's origin and the composite's origin. If not present, it is undefined and the importing system is free to set it arbitrarily.

<NODE>

This is the meat of the composite. See above for a description of this data structure.

Standards for hypertext source files: the experience of UNIX Guide

P.J. Brown
Computing Laboratory
The University
Canterbury
Kent, CT2 7NF
England

In real-world applications, it is rare that a hypertext system provides a complete solution. Instead the solution normally comes from a combination of a hypertext system with other tools. Thus, as Meyrowitz (1987) has argued in his powerful position paper "The missing link: why we're all doing hypertext wrong", one of the most desirable attributes of a hypertext system is that it should fit easily into its environment, and allow a close interaction with other tools in that environment.

There is now a movement towards standardisation in hypertext systems, in particular a proposal that source files for hypertext systems should follow a standard form so that material can be interchanged between different systems. The market forces pushing this standardisation effort are obvious, but we must ensure that new standards do not detract from the interaction between hypertext systems and other tools. At an extreme, a standard that made it easy for a hypertext system to exchange files with other hypertext systems but hard to exchange with anything else would be a disaster.

Do we use text-files?

Choosing a file format for hypertext systems is similar to choosing a file format for word-processing systems. Indeed many hypertext systems support a good repertoire of word-processing operations. Hypertext systems have the added needs of representing hypertext constructs and links. Hopefully any standard will encompass all documents, irrespective of whether they are created from word-processing or hypertext. For hypermedia systems, similar considerations apply to the other media, but this paper concentrates mainly on text.

A basic choice is whether files should be a *text-file*. By a text-file we mean a linear sequence of text with embedded mark-up but with no embellishments such as file-headers, associated tables, embedded pointers, etc.

This paper argues the advantages of text-files. The argument is based on experience with the UNIX implementation of Guide, which uses a text-file format. Most of the material is concerned with nitty-gritty practical experience rather than with any underlying theory, but standards cannot ignore these practical aspects. We shall start by emphasising the properties of UNIX Guide that influence its file format.

UNIX Guide

A central aim of the UNIX implementation of the Guide hypertext system is that it should fit well into a UNIX environment (Brown, 1989). Indeed it is this facet, more than anything else, that has caused UNIX Guide to be different from the implementation of Guide marketed by Office Workstations Ltd (OWL) which runs on Macintoshes and PCs. OWL Guide successfully fits into its environment, which is very different from UNIX and has a strong house-style that pervades most of the software that runs in that environment.

UNIX Guide — and henceforth all references to Guide should be taken as UNIX Guide — tries to follow the original UNIX 'Small is beautiful' philosophy, though this philosophy has perhaps been weakened over the years to the less catchy 'Medium-sized is beautiful'. Guide cannot hope to provide all the facilities that users may want. Instead it should be good at one thing, hypertext, and use other tools to provide functions that they are good at.

Characteristic features

Every hypertext system has some characteristic features that set it apart from the herd. In the case of Guide there are three such features: UNIX orientation, which we have just discussed, late binding and the scroll model.

Guide's late binding philosophy is that fixing of hypertext links should be delayed to the last possible moment; this is normally at run-time when the link is selected for the first time. Late binding has a number of benefits, arising from the dynamic nature of links.

The Guide author specifies a link by a symbolic name (e.g. 'Lesser-spotted woodpecker'). If the link goes outside the current file a filename is appended to the symbolic name (e.g. '... in /x/y/z'). The destination of a link is a Guide 'definition' with the same symbolic name as the link. When links are saved in Guide source files they follow this symbolic form — they are just a sequence of characters attached to the button-name that is the source of the link, and only at run-time do they cause a link to be forged (by searching for a definition that matches the given name). Late binding is therefore a force that makes source files simpler and flatter.

The third characteristic feature of Guide is its scroll model. A Guide document is a continuous scroll, and when buttons are selected they are replaced in-line by the corresponding button-replacement, thus causing the scroll to grow and shrink as buttons are selected/deselected.

Groups of buttons can be combined into larger units, called *enquiries*. In Conklin's (1987) terminology an enquiry is a *region*, which is replaced if any button within the region is selected. In page-based systems that have a single current page, e.g. HyperCard, the region to be replaced is always the whole current page. Enquiries offer more flexibility: in particular, at one extreme they can be made to encompass the entire current document. If this is done, Guide, notwithstanding its underlying scroll model, can be used to simulate these page-based hypertext systems. (See Brown (1990) for a discussion of a large application that takes advantage of this.) At another extreme the region of replacement can be made null: everything remains; if, in addition, a button is made to throw its replacement up in a new window (as Guide 'action-buttons can be made to do) instead of in place of the original button, then the end result has the flavour of NoteCards. Overall, therefore, the scroll model is not fundamentally different from a page-based one.

Nevertheless the scroll model, with in-line replacement the norm, has influenced the source file design. For the simplest type of button, which has a fixed replacement that is associated with that button and no other, the button-replacement comes immediately after the button-name in the Guide source file. This simplest type is button is also generally the commonest, since it is used in hierarchical expansions.

Guide source files

Having covered Guide's characteristics we can now describe its source file format, and the advantages that come from using such a format.

As we have said, the file format is that of a text-file: a sequence of text and graphics with embedded mark-up. The mark-up simply shows where Guide constructions (e.g. buttons, replacements, enquiries, 'ghosts' — Guide comments) begin and end. All the necessary information is carried by this mark-up: there is no file-header and there are no associated tables, etc.

The mark-up follows the format of *troff* requests. For example, a button-name 'Lesser-spotted woodpecker' would be represented as

> . Bu *button-attributes*
> Lesser-spotted woodpecker
> . bU

Thus the *Bu* and *bU* requests mark the beginning and end of a button name, and the *Bu* request has as its argument a description of the button's attributes. (For better or for worse, attributes do not figure strongly in Guide and the *Bu* request is, in fact, one of the few Guide requests that has attributes.)

The purpose of this paper is not, of course, to propose *troff* format as a standard. As far as Guide itself is concerned it would be equally easy to replace the *troff* syntax with any other syntax that had mark-up embedded in the text, e.g. our previous example could have been in the SGML (ISO, 1986) form:

< Button ... > Lesser-spotted woodpecker < \Button >

However, given the need to use other UNIX tools, the use of *troff* syntax, which is a UNIX standard, has certain advantages. For example:

- *spell*, the UNIX spelling checker, can be used on Guide files without any adjustment. (It automatically strips off *troff* mark-up by using the *deroff* utility.)
- if Guide files are to be formatted and printed on paper, *troff* can do the job. For example the *Bu* request can be made a macro which, inter alia, switches to bold-face so that button-names come out in bold. (The names of Guide requests have been deliberately chosen not to clash with other *troff* requests.)

These UNIX-dependent advantages of Guide's mark-up should not, however, be over-emphasized, and if SGML-based tools had been readily available SGML format would have been a better choice.

Readability

The majority of Guide users are unaware of how its source files are stored. However some authors do need to look at or to generate source files, and for them it is a huge advantage that the files are fairly readily understood by humans. Indeed the very first Guide implementation (1984–5) had a file format involving esoteric binary codes, and perhaps the greatest step forward in Guide's development has been the banishing of this mumbo-jumbo. Sample benefits of the readable form are:

- it can be edited using specialist editors. Although Guide offers editing, this is not its forte; elaborate editing, e.g. global replacement of a pattern, can be done by a tool that is specially designed for such tasks.
- it makes conversion programs easier to write and debug, a point we discuss later.

Other media

Although this paper concentrates on text, since we believe it will predominate in most hypertext applications for the foreseeable future, it is not sensible to ignore other media. They can be either:

(a) stored in separate files, whose names are referenced in the main text-file. These separate files would hopefully be represented in the appropriate standard form for the media.

or (b) embedded in the form of comments in the text-file. Often the content of these comments will appear as arbitrary binary codes, sanitized if it is necessary to avoid 'difficult' codes such as end-of-file and end-of-line.

UNIX Guide offers both. If the second approach is used a bit-map picture is represented as:

```
.Pi
.\" bytes representing binary encoding
.\" bytes representing binary encoding
  .
  .
  .
.pl
```

Each line of the binary encoding is made to appear as a *troff* comment. This is important, as it causes utilities such as *spell* to ignore these lines; otherwise there could be spurious reports of spelling errors.

In order to create the encoding of a picture, Guide has to capture the raw picture in the first place. (The raw picture will typically have come from a drawing program or a scanner.) Like most other software, Guide tries to avoid input modes ('This is a picture', 'This is a text file'). Input modes can be avoided if files have a type associated with them. UNIX has a somewhat basic — unkind people would say crude — mechanism for attaching a data type to a file. This is the 'magic number'. It helps Guide avoid input modes though it becomes difficult if material comes in through a pipe rather than direct from a file. Overall a standard could not assume that every file system provides a satisfactory mechanism for attaching a data type of a file. Hence if source files are represented in a wide variety of forms, corresponding to different media standards, the user will sometimes be forced into the use of different input modes.

Aims of standards

It is worth pausing at this point to consider the purpose of hypertext standards. Three important aims of hypertext standards should be:

(1) to allow import/export of documents, or more generally to allow sharing of documents with other software.

(2) to allow exchange of documents with other hypertext systems.

(3) to allow existing tools to be applied to standard documents.

The last of these is often overlooked, but if there are no tools associated with a standard the standard will be a standard that no-one uses — a bitter lesson that many have learned. In most environments (and especially in UNIX) the vast majority of existing tools use a linear textual format. This may be a sad commentary on the state of the world, but it is the reality. Hence choice of a text-file format as a standard has big advantages.

One can argue on the relative importance of (1) to (3) above. Personally we rate (1) and (3) equal, with (2) far behind. We shall now discuss (1) further.

There are two sub-cases of (1). Firstly there is the import/export case where material produced by another tool is converted to hypertext form or the hypertext form is converted for use by another tool. The other tool may be a word-processor, a database, a programming language compiler, a drawing tool, etc. Secondly there is the utopia which the standard envisages: all material shares the same format and no conversion is necessary — though several problems remain, as we shall see later.

Conversion may be done in advance or on-the-fly. The latter is, of course, preferred if conversion is a fast process, since it does not involve keeping two separate documents up to date. Conversion is normally a dreary and unsatisfactory process, but there are three ways in which the hypertext file format can help:

- a simple textual format facilitates conversion.

- it helps if hierarchical buttons have their replacement immediately following. For example it then requires only a trivial effort, when converting a word-processor file, to map section headings into button-names and the body of the section into the button's replacement.
- a format that is readable by humans aids the debugging of conversion utilities. (Sadly, conversion utilities, because of their ad hoc nature, tend to take a long time to debug. Each new source document brings a new crop of problems.)

Pipes

If conversion is performed on-the-fly the UNIX pipe — now available, in one form or another, in most operating systems — is a convenient way for transferring data. Hence Guide is frequently used as a component of a pipe.

Following the general UNIX philosophy Guide does not know or care whether its input comes from a source file or a pipe and the same format applies to both.

In this environment the following characteristics of source files have proved valuable:

- source files are text-files — again this advantage comes first: most piping mechanisms are based on the stream-of-characters model.
- a text-file containing no mark-up at all is a valid source file. Such material (e.g. the whole or part of existing non-structured files) is commonly used in building Guide documents and does not, therefore, require a special input mode.
- a concatenation of source files is a valid source file. Moreover a source file can be *included* within another. Thus a utility such as the C pre-processor can be used to build the Guide input from a combination of existing source files. (These may, indeed, be parameterised using pre-processor statements such as *define* and *ifdef*.)

Newlines

A small issue of some importance is the treatment of newline characters, and in particular whether they should be hard or soft. Since newlines are hard in ordinary text files, Guide generally treats newlines as hard. However a newline that precedes a Guide request is ignored. (A newline preceded by a null Guide request therefore acts as a soft newline. When Guide saves a file it inserts a soft newline if an output line is getting too long — very long lines knock out many UNIX utilities.) Obviously, when material is imported or exported, soft newlines and other soft mark-up needs to be stripped out before transmission.

Dynamic interchange

Ideally a hypertext system should support a dynamic interaction with its environment. Thus data should be shared with other programs while the hypertext system is running. It is natural that the source file format applies to such data as well as to data that is pre-stored in source files. In Guide, the selection of a button can cause a program to be run, and the output from that program serves as the replacement of the button. This output follows the normal Guide source format; usually it is a sequence of ASCII characters without any mark-up. Sometimes, however, the output may involve hypertext structure: for example in one application, a button launches a program that is a retrieval system. The program searches for a given term and converts the hit list into a hypertext structure that makes it easy for the user to examine the hits that seem most relevant. This structure is duly displayed by the hypertext system. In another application a button runs a program to produce a report of items currently in stock, and this output is produced in a hierarchical hypertext format.

The issue of standardization also affects the programs that are executed within hypertext systems. Most systems contain their own programming language, and in HyperCard this is a major part of the system. However experience suggests it would be hopeless to expect every hypertext system to abandon its current programming language and adopt a new standard one.

Saving

The 'save' operation from a hypertext system may involve:

(1) saving what is seen.

(2) saving what is seen, together with the hypertext structure behind it.

It is (2) that interests us here, since it creates a hypertext source file. This output file need not relate directly to a single input file: at one extreme it could have resulted from loading several input files and editing them; at the other, the material saved could be a small fragment of an original input file.

Cut-and-paste, when used to cut *from* the hypertext system, is a special case of saving. Ideally both (1) and (2) above should be offered, though Guide currently only offers (1). Case (2) is useful if the material is to be pasted back into a hypertext document.

Saving may go directly to a file or into an output pipe.

Saving presents no problem if source files use a text-file format. If the source format involves file-headers or the like, it requires more thought and perhaps more user action, particularly if the original input came from diverse sources.

Sharing files

Earlier in this paper we wandered in the anarchical world of conversion programs; it is now time to move on to the relatively utopian idea of *sharing* information so that an identical file can be processed by many different systems.

Let us assume that two programs X and Y share the same file. (X and Y may be different hypertext systems or one or other of them may be, say, a word-processing system.) A user of X may load the file, edit it and then save it. Clearly the file should still be usable by Y.

This apparently simple requirement requires care. Inevitably there will be some operations Y can do, but X cannot. Assume for example that Y can display text in different point-sizes but X cannot. If a file contains mark-up indicating a change of point-size X must preserve this information when a file containing point-size changes is loaded into X, edited and subsequently saved. As a greater challenge, X must behave sensibly when editing involves material that contains point-size changes: what happens if half of a string in a large point-size is copied, and the instruction to increase the point-size is copied but the corresponding instruction to set it back is not copied?

Guide currently makes an attempt to deal with these issues. It has an experimental system for sharing files with *troff*. If a *troff* file is loaded into Guide, Guide tries to take account of mark-up it can handle, e.g. new paragraphs; other mark-up, such as change of point-size, is ignored. However all the original *troff* mark-up is loaded into Guide in the form of 'ghosts' — comments that are only visible to Guide authors, not to Guide readers. When a Guide file is saved, these ghosts are converted back to the original *troff* mark-up, thus re-creating the original file. Given that Guide authors can see these ghosts, they will, hopefully, be aware of the implications of the mark-up when they perform edits.

On the other side of the sharing, when *troff* is using the file, there are fewer problems, not least because *troff* has no save operation. It is, in this situation, a happy property of *troff* that it completely ignores requests it cannot recognise; thus Guide mark-up is ignored.

Overall the current Guide sharing system just about works, but could profitably be replaced by something built on sounder foundations.

Errors

If source files may be generated by conversion tools, editors, etc., they may well contain errors. The design of source files should therefore contain enough redundancy for such errors to be detected. The design should also bear in mind that, on detecting an error, the hypertext system should have sufficient information to give a decent error message and stop gracefully, retaining as much of the source file as possible.

Abstractions and discipline

The focus of this paper has largely been on the present rather nasty world. Ideally standards should look to the future as well as covering the present.

Current usage of Guide (and doubtless of other hypertext systems too) has shown up two deficiencies:

(1) a need for higher level abstractions than links, which are gotos.

(2) a need for each application to evolve a hypertext house-style and to impose this.

The two needs are related: many aspects of a house-style can be imposed by designing some special abstractions and then ensuring that authors use *only* those abstractions. This is similar to the way that document standards such as ODA (ISO, 1988) and SGML impose a general document architecture.

The ICL Locator project (Meehan, 1987; Brown, 1990), one of the biggest current Guide applications, has successfully tackled (1) and (2) by producing a preprocessing tool for Guide that helps (and constrains) authors to produce the required Locator style. However preprocessors are not always the answer for the same reason that preprocessors to compilers for programming languages are not always the answer. In the latter case the program author, when maintaining/debugging a program, usually needs to be aware of its intermediate form and thus the power of the abstraction that the preprocessor provides is lost.

Experience also shows that some environments want discipline and some want freedom. Thus heavyweight mechanisms that affect everybody need to be avoided.

Overall, therefore, it is desirable that source file formats contain facilities for defining or imposing abstractions, but these should be optional. It should still be possible for draconian managements to enforce their requirements; for example, currently some managements do not release the real Guide to their authors, but equate 'Guide' to a UNIX shell-script which loads the real Guide with certain options already pre-set, and perhaps with some of the items in Guide's normal menu either suppressed or replaced. (Guide options are, incidentally, mostly controlled by UNIX environment variables and switches; some could profitably be controlled by mark-up within source files, but currently this is not supported.)

Size of file

The design of source file formats is somewhat influenced by the size of a typical file: is it a single 'page' or could a whole encyclopedia be stored in a single file. In practice Guide authors vary considerably: some have tiny files and some have files containing megabytes of text. In the latter case there is a significant pause while the file is loaded but thereafter speed is superb.

Typically the initial screen consists of a summary, which consists of a skeleton document with buttons representing the components of the document. Initially no buttons are expanded. However Guide's source file format, where normally the replacement of a button immediately follows the button-name, means that the whole source file needs to be loaded in order to paint the initial screen. Indeed because of this Guide always loads complete source files, making no effort to restrict itself to the parts that are actually needed. In the environment where Guide runs, workstations with a lot of real storage, supplemented by virtual storage, this has caused no problems. However OWL's Guide, which can run in much more constrained environments than UNIX Guide, has adopted a file format that does allow parts of the files to be loaded. OWL uses a structured file format where associated tables designate where constructions begin and end.

Conversion between hypertext systems

Although UNIX Guide and OWL Guide have identical parentage and similar hypertext mechanisms, it would be a major job to convert source files between the two. This is not because file formats are different, but because there are significant differences in the way linking is done (e.g. UNIX Guide's late binding approach is not found in OWL Guide).

A conversion has never been attempted but, if it were, it would be a similar exercise to converting between two somewhat similar programming languages: you may get an automatic tool to convert 90% of a program, but the rest would need doing by hand. Even within the 90% that was automatically converted, there would be odd differences in program behaviour.

A complete conversion between two radically different hypertext systems would clearly be harder still. It is not the source file format that is the problem, but fundamental differences in approach. This is why we believe that this is the area where standard file formats have least to offer. There is, of course, the possibility of a deeper standard which specifies how hypertext systems actually work. In practice there is, however, no more chance of getting creators of hypertext systems to agree than getting designers of, say, programming languages to agree.

Conclusions

The tone of this paper has been at least lukewarm about standards.

Nevertheless UNIX Guide can hardly claim to be a major force that will materially affect that standardisation movement, and hence standards may come. If they do come we hope they:

- are geared to exchange with other software (word-processors, picture drawing programs, databases, etc) rather than specifically with other hypertext systems.
- are geared to taking advantage of existing tools.
- are based on ASCII files that can be read, edited, etc, by humans, and can be sensibly transmitted down pipes and similar mechanisms.
- can treat straight text files as a subset of hypertext files, rather than as special cases.
- are not based on a specific linking mechanism. If late binding is used, the linking mechanism is not very relevant to source formats.
- allow flexibility in the region of replacement so Guide enquiries and their equivalents in other systems can be supported.
- cater for higher-level user-defined abstractions and house-styles.
- allow other software to *share* hypertext files without the need for conversion problems.

References

- Brown, P.J. (1989). 'A hypertext system for UNIX', *Computing Systems*, **2**, *1*, pp. 37-53.
- Brown, P.J. (1990). 'Hypertext: dreams and reality', in Lennox, G. (Ed.) *Hypertext/Hypermedia and object-oriented databases*, Kogan Page, London.
- Conklin, J. (1987). 'Hypertext: introduction and survey', *IEEE Computer*, **20**, *9*, pp. 17-41.

BEST COPY AVAILABLE

- ISO (1986). *ISO 8879 — Text and Office Systems — Standard Generalized Markup Language (SGML).*
- ISO (1988). *ISO 8613 — Text and Office Systems — Office Document Architecture (ODA) and Interchange Format.*
- Meehan, D.P. (1987). *Locator: a system for service-desk 8801 fault diagnosis*, M.Sc. thesis, Kingston Polytechnic, Kingston, U.K.
- Meyrowitz, N. (1987). 'The missing link: why we're all doing hypertext wrong', position paper, *Hypertext 87*, University of North Carolina.

Standards: What Can Hypertext Learn From Paper Documents?

Fred Cole
Heather Brown

Computing Laboratory
University of Kent
Canterbury
CT2 7NF
England

1. Introduction

Hypertext literature tends understandably to concentrate on what is new and to ignore, or take for granted, the properties of hypertext that are also present in paper documents. The purpose of this paper is to consider how the expertise that exists in standards and models for paper documents can be used to save effort when designing a standard for hypertext, and how to make hypertext and paper document standards compatible. Section 2 discusses some relevant similarities between paper and hypertext documents. Section 3 introduces relevent aspects of the Office Document Architecture (ODA) [1] and suggests ways to build on ODA to create a standard that combines the strengths of the two areas.

2. Similarities between paper and hypertext documents

2.1. The need to separate the logical structure and its presentation

Although hypertext systems vary widely in appearance and functionality they generally have similar underlying document structures — directed graphs in which the nodes hold the content and the arcs represent links of various types. The way in which the nodes and links are presented on the screen, and what happens when a link of a particular type is activated, are peculiar to (and usually hardwired into) the hypertext system.

If a standard for hypertext is to be effective, it must allow a hypertext to be created on one system and presented on another. In particular it must allow for the possibility that the receiving system does not have the capability to perform the presentation as intended on the original system. To do this it should represent separately:

(i) the components in the underlying logical structure;

(ii) the specification of presentation facilities on each participating system (including dynamic properties such as the actions allowed when hotspots are selected);

(iii) a mapping from (i) to the relevant set in (ii) for each participating system.

This separation of the logical structure from the method of presentation is not just an inconvenience needed for portability; it is a positive feature that can be used to give hypertext some of the advantages that were given to paper documents by generic markup and structured editors.

Markup of documents intended for paper used to be, and in many cases still is, presentation oriented. Formatting commands are inserted into the document to request explicit presentation features such as moving the current print position or changing to a given font style or size. Generic markup, on the other hand, is concerned with the logical structure of the document — it marks portions of the content as belonging to particular named classes. The actual layout and presentation are bound to the name later (either by the publisher, using traditional markup, or by a computer formatting system). Generic markup is essentially for non-interactive systems. The interactive equivalent is the structured document editor, which works in a similar manner by assigning a named class to each document constituent and providing separate 'style sheets' to specify the presentation of constituents belonging to the class. The appearance of all constituents belonging to the class can be changed by altering the style sheet.

In both cases the effect is to separate low-level presentation details from the logical document structure and content (as in (i) and (ii) above) and to allow or provide a means of binding the two together at a later stage. This late binding corresponds to the mappings in (iii) above.

In the logical structure of the document the named classes should correspond to the function of the content rather than the method of its presentation ('title' or 'reference' rather than 'change to bold type', for example). Generic markup and structured editing are acknowledged (see [2] for example) to have many advantages including:

- making it easier to present the document in another style (that of a different publisher, for example) without extensive manual changes to the text — this is the paper equivalent of presenting a hypertext on a different system.
- helping to maintain a consistent style throughout the document, and making it easier to enforce a house style.
- improving typographic quality by discouraging authors from dabbling in low level details and leaving the design of styles to experts
- forcing the author to consider the structure of the document. This usually results in a better structure — and could be particularly important for hypertexts.

Where layout and presentation facilities are complex, this separation of the logical and presentation aspects of the document often results in considerable factorisation of information and consequently in reduced costs for transmitting a document.

2.2. Links

Paper documents have links — intra-document links to components of the logical structure ("see section 3.5") or to part of a particular representation ("see page 27"), and inter-document links (bibliographic references). Each link (in a well-written document) is accompanied by some indication of what the reader can expect to find at the other end, or at least the reason the author has for directing the reader there. Hypertext differs only in that, instead of indicating the position ("page 27") of the remote object, it offers some means of automatically accessing and presenting the remote object.

If a system is to be able to edit or reformat a paper document and still retain the integrity of its links, then each link must be represented at the logical level in much the same way as it would be in a hypertext. It might, for example, have a type, a reference to the identifier of a remote object and, associated with the type, a specification for how it is to be presented.

2.3. Hierarchical structures

In paper documents the logical components referred to above are typically arranged in a hierarchical tree-like structure. A book, for example, might contain chapters which contain sections which contain paragraphs. This structure is primarily a tree but it may be supplemented by link components that cut across the normal tree links and turn the structure into a directed graph.

Although hypertext systems emphasise the links more than paper documents, their underlying models are similar. Indeed, several hypertext systems recommend or enforce a general hierarchical model to minimise the well-known problem of readers becoming lost [3,4].

To represent a hypertext within the hierarchical model for paper documents, we could start by assuming that the logical structure components referred to above might simply be the links and nodes of the document. In this case each node would be very simple, consisting of a single piece of basic information together with hierarchical and non-hierarchical links. The hierarchical links would form the basic tree structure, and the non-hierarchical links would be the link components.

Most hypertexts could not be represented by such a simple structure, however, and there is a need for internal structure for a node. A finer granularity is needed, in which each node is structured hierarchically into a number of subordinate components (including links) representing paragraphs, parts of paragraphs, diagrams, buttons, hotspots and such like. The hypertext node thus becomes a subtree and this makes it possible to represent the node in a way very similar to that in which we represent a page of a paper document (although in some cases the 'page' might be so large that it needs to be scrolled). Rules for laying out and presenting the components of the node could then be specified in the way they are specified for a page of a paper document.

2.4. Style and the problem of getting lost

As shown above a single node of a hypertext is similar in many respects to a page or logical section of a paper document, and it has long been recognised that the meaning of a page of information — and the ease with which this meaning is understood — is very dependent on the skill with which the page is laid out. Those unskilled in the art of typography are well advised to leave the design of the document styles to experts. For a hypertext, style would include the positioning and presentation of different types of button or hotspot as well as text and diagrams. The structures described above would allow all the sophistication used for laying out a page of a paper document to be applied equally to laying out a node of a hypertext.

Early applications of the standard will probably be in automatic translators between existing hypertexts and the standard, in which case the separate logical structure and the late binding will initially be hidden from end users. It would be wise however to ensure that the standard allows for future improvements in hypertext. A reasonable assumption is that hypertext systems could learn design techniques from paper document processing systems, including the principles inherent in generic markup, in order to gain the advantages listed above and especially to help authors to improve the styles of their hypertexts.

Well defined and consistent styles have a bearing on the problem of getting lost in hyperspace [4], the solution to which has often been considered to be a matter of giving the user a suitable overall graphic view (or map) of all or part of the document. There is reason to believe that this may not be the only or even the best method [5] and that perhaps good authorship may make it unnecessary for the user (including authors?) to be aware of the underlying directed graph. Well-designed generic styles could be a way of helping users with this problem.

2.5. Compatibility between paper and hypertext documents

It would be foolish to ignore the need to produce a paper version of part of a hypertext, and it also seems sensible to make provision for readers to have the advantages of hypertext navigation when viewing a document on the screen — even if the document is eventually intended to be read from paper. These aims could best be achieved by having a common underlying representation for the structures of both types of document, together with well designed ways of mapping those structures onto different forms of representation. It is not suggested, of course, that a document designed for paper would necessarily make a good hypertext or vice versa, only that a usable representation should be readily available by applying different presentation styles.

3. A hypertext standard based on ODA?

ODA is a standard for the storage and interchange of complex multimedia documents. The ODA document model is hierarchical and object-oriented. It caters for both source (processable) documents and output (formatted) documents. Currently ODA documents can contain three types of content (character, raster graphics and geometric graphics) but other types of content will soon be added.

Several major extensions to ODA are already under consideration in the relevant committees and working groups. These include tabular layout, video material, the inclusion of data in documents — and hypertext. The SGML [6] community is also starting to consider hypertext extensions. It would be tragic if three separate hypertext standards emerged: one based on ODA, one based on SGML, and a completely separate one from the hypertext community. After several years of rivalry and backbiting the ODA and SGML committees are showing encouraging signs of working together, so there is some hope that these two may merge.

The details and suggestions given below are based on ODA, largely because ODA currently includes graphics and images and defines a layout process to map from the logical structure of the document to a formatted form. However, the general principles could apply to SGML when used with DSSSL [7] which defines a presentation model for SGML documents.

The following subsections describe the features currently in ODA that make it useful as a basis for a hypertext standard and then the features that we believe must be added. These new features are needed to improve the ability of ODA to represent all the features of high quality paper documents, but are also intended to prepare the way for the hypertext extensions to ODA.

3.1. What ODA can already offer to hypertext

The following sections give a brief description of ODA as it applies to paper documents.

3.1.1. ODA Document Architecture

ODA provides a tree-like model of a document. The *structure* of the document is given by the shape of the tree, while the *content* is stored entirely in the leaf objects. *Attributes* provide information about the objects. A few of the most important attributes are introduced in the examples and discussion below. Only one needs to be mentioned at this stage. This is the *content architecture* attribute that defines the type of content for each leaf object and thus allows different types of content to co-exist within the document.

An ODA document is described by two structures. The *logical* structure divides and subdivides the content of the document into logical objects that mean something to the human author or reader. A logical object may be a general item like a section, title, paragraph or reference. Alternatively it may be a specialised item like a telephone number or price, or a collection of related information like a list of companies selling a particular product. Only the lowest level objects, such as titles or prices, have content.

The *layout* structure is concerned with a visible representation of the content. It divides and subdivides the content into page sets, pages, and rectangular areas within pages. Rectangular areas with nested areas defined within them are known as *frames*. The lowest level areas are known as *blocks* and, by definition, are the only areas to have content associated with them. A frame might be used to represent a column of text, for example, with nested blocks for the content of individual paragraphs.

Each document has its own specific logical and specific layout structure, but their creation is guided and controlled by *generic* document structures for that particular class or 'style' of document. These are sets of object type definitions (one set for logical objects and one for layout objects) that specify the types and combinations of objects allowed. In ODA terminology the definitions constitute the generic logical and generic layout structures for a document class.

3.1.2. Examples of ODA Structures

This section illustrates the structures introduced above by presenting snippets of the generic structures that might be used for a journal containing technical papers. It also introduces a few important attributes.

The generic definition for each non-leaf object has an attribute called *generator for subordinates* that describes how the object may be made up from subordinate objects. These indicate that subordinate objects may be optional (OPT), required (REQ), repeated (REP), or optional and repeated (OPT REP), and that a group of objects may occur in a given sequence order (SEQ), in any order (AGG), or as a choice where only one of the group occurs (CHO). The information given in these attributes provides a simple grammar for the primary structure of the document class.

Figure 1 shows the generic logical structure for a single technical paper in the journal. It indicates that the paper consists of a compulsory title, followed by a compulsory author's name, followed by an optional abstract, followed by one or more sections. If the abstract is present it consists of a single paragraph. Each section begins with a subtitle. The 'REP CHO' construct indicates that the subtitle is followed by a series of paragraphs or lists occurring in any order. Lists consist of one or more list items. (In practice, a more complex structure catering for items like footnotes and diagrams would be needed.)

The corresponding generic layout structure might define one page style for the first page of the paper, and a different style for all subsequent pages. Figure 2 shows the top level of such a structure. The 'Title page' contains a 'Header frame' representing an area set aside for the title, author's name and abstract, and a 'Body frame' for the start of the first section. The 'Continuation pages' contain 'Continuation body frames' to hold the rest of the sections. (Again, in practice, further frames would be needed for items like running titles.) Blocks are not included in the generic layout structure but are assigned to pages and frames during the layout process as outlined below.

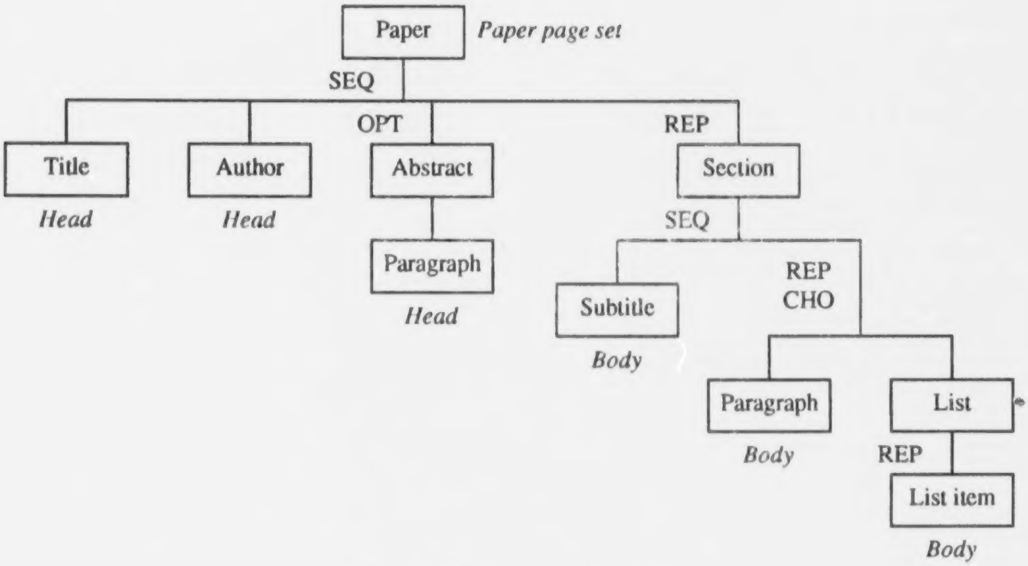


Figure 1: Generic logical structure

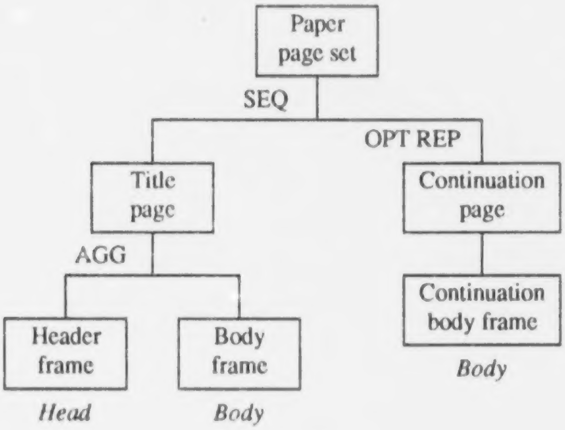


Figure 2: Generic layout structure

ODA's *layout process* decides exactly where each item of the document is to be placed. It uses the specific logical structure, the generic structures, and the content architectures to create the specific layout structure. It works at two levels

- *Content layout* takes portions of content and lays them out into blocks. This stage is dependent on the content architectures involved and on sets of attributes known as *presentation styles*.
- *Document layout* places blocks in frames or pages. This stage is dependent on sets of attributes known as *layout styles*.

The content layout process thus deals with character sets and the fine positioning of items within blocks, while the higher level document layout process decides how to place the blocks within pages and frames.

The document layout process is guided by three attributes whose values are shown in italics in Figures 1 and 2. *Layout object class* is normally used to indicate that a major logical division of the document should be directed into a particular page or page set. In the example the logical 'Paper' has its *layout object class* defined as 'Paper page set'. This dictates that each paper must be laid out in a single instance of the page set shown in Figure 2.

Within a layout object class, the attributes *layout category* and *permitted categories* can be used to direct logical objects into different frames. If a leaf logical object is given a layout category name, it can only be laid out in a frame that has the same name as one of its permitted categories. In the example the only category names used are 'Head' and 'Body'. When the layout process tries to place the blocks corresponding to the title, author's name, and abstract (if present), it will look for a frame with 'Head' as a permitted category, and will therefore create a 'Title page' and place them in the 'Header frame'. But when it reaches the blocks corresponding to the contents of the sections it looks for frames with 'Body' as a permitted category, so it uses the 'Body frame' until that is full and then creates 'Continuation pages' as necessary in order to use the 'Continuation body frames'.

When the specific layout structure has been created, it associates the document content with pages, frames and blocks. The two specific structures are related and come together at the level of the content. Figure 3 shows a fragment of the specific structures for the beginning of a paper. It assumes the paper has no abstract and that the first section begins with three paragraphs, only one of which fits onto the title page. Figure 3 shows a neat one-to-one correspondence between logical objects and layout objects. This often occurs, but not always. Logical content portions may, for example, be split between blocks (when paragraphs are split over pages) or concatenated into paragraphs occupying a single block.

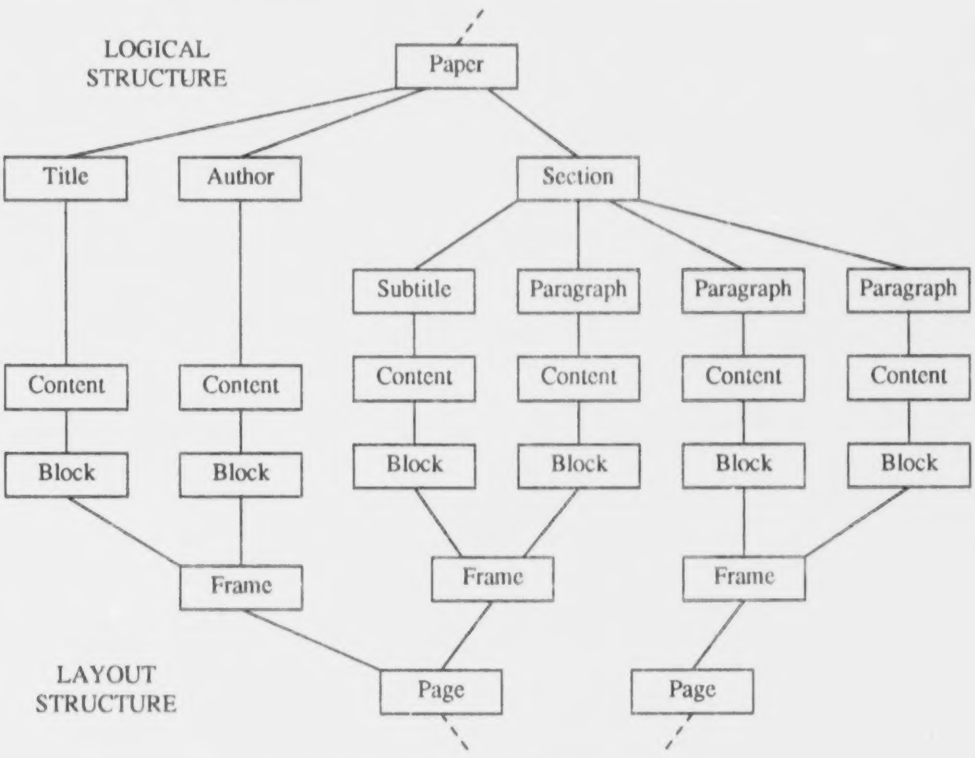


Figure 3: Specific logical and layout structures

3.1.3. Providing Different Views of an ODA Document

The previous section gave only a brief sketch of the ODA layout process, but it should be sufficient to show that the appearance of a specific logical document can be altered by judicious changes to its generic layout structure. As a simple example, deleting the 'Body frame' from the 'Title page' in Figure 2 would cause each paper to be laid out with only the title, author's name and abstract on the first page. There would be no frame on the first page with 'Body' as a permitted category, so the first section would have to start on a new page in a 'Continuation body frame'.

More radical changes to the layout can be achieved by altering the attributes that make up the layout and presentation styles. The attributes in these styles apply to logical objects, but the objects contain only the identifier of the appropriate style. The styles themselves are held separately. This provides a more concise document representation and allows the styles to be changed without changing the logical structures.

The layout styles include the *layout object class* and *layout category* attributes (described in the previous section) and other attributes governing the selection of frames and the positioning of blocks within a frame. The *same layout object* attribute, for example, constrains the block containing the logical object to share the same frame as the block containing another specified object, while *new layout object* constrains the block containing the object to start a new frame. *Offset* and *separation* control the minimum spacing between adjacent blocks, and the relative position of blocks is dictated by *fill order* which allows normal top-to-bottom positioning or traditional footnote positioning.

The presentation styles guide the lower-level content layout process and thus affect the appearance of content within individual blocks. They contain different attributes for different content architectures. For character content, for example, they include attributes affecting the indentation of the first line, the distance between lines, and the initial font size.

Changing the generic layout structure and the styles can lead to significantly different views of the same logical document. Page and margin sizes can vary, single or double column layout can be used, and paragraph spacing and font size can change. In particular, it is possible to cater for different 'house styles' by this means and to provide different styles for interactive editing and the final printed version. ODA is not as flexible as it should be in this respect because it has insufficient separation between the logical and layout structures. We are attempting to get this changed (see below).

3.2. What ODA still lacks

The structures and styles introduced above form a good basis for a flexible standard for paper documents and provide at least some of the requirements for a hypertext standard. We have identified a number of deficiencies in the ODA standard and have investigated changes to the standard that would overcome them. The changes are needed in order to improve the representation of paper documents but were designed with the aim of preparing the way for an extension of ODA to deal with hypertext. ISO/IEC JTC1/SC 18/SWG (the special working group responsible for changes to the standard) has already declared its intention to develop such an extension. We have explained the deficiencies and our suggestions for improvement in a paper [8] that is to be considered by the special working group in January 1990. Brief outlines of the deficiencies for which we have offered cures are given below.

3.2.1. *Separating logical structure from presentation*

One of the strengths of ODA is its attempted separation of the logical and layout structures, but this does not go far enough, so we have made suggestions to make it complete. If it is required to change the style of a document (to the house style of a different company or different publisher, for example) it should not be necessary to edit the logical structure, only to apply a different set of layout and presentation styles to create a different "view" of the same logical document. This facility to change the view without changing the document is part of the answer to the problem of exchanging hypertexts between different systems that have different presentation capabilities or different presentation conventions.

3.2.2. *Comprehensive attribute inheritance*

The ODA mechanism for inheriting layout and presentation attributes, in spite of its complex algorithm for finding default values, is insufficient. If an attribute value is not specified for the object or its class then the value can only be inherited according to the object's position in the tree and not according to its class (chapter, list etc.). Our suggestion for supplying this facility is the addition of 'style tables' as described in [8]. The use of style tables enables the style inherited by an object (and therefore the way it is formatted) to depend both on its class and on its position in the document. This mechanism is valuable for hypertext representation, making it possible to distinguish objects of the same type that are in different states (open and closed buttons for example) and can be extended so that it can specify changes of state (such as those that take place when a hotspot is selected) by changing the style table.

3.2.3. *Links*

In both paper and hypertext views a document designer must be able to specify the purpose of each link, and to specify how the layout process can express that purpose. In this respect the requirements for links are very similar to those for logical objects, so it seems reasonable to deal with them in the same way — by having classes for links. The class of the link should determine how and where in the document the link can be used, and it must be possible to specify the representation of the link in a way that depends on both the class of the link and also its position in the document.

We discovered that a small number of additions to the definition of ODA logical objects allows a document designer to use those logical objects as links, with all the functionality described above. These additions do not in any way change existing definitions or change the validity of existing documents.

3.2.4. *Selective and multiple presentation*

ODA does not have a mechanism for specifying that a logical object should be ignored in the layout process, nor that it should be laid out more than once. A facility to ignore objects could, for example, allow a document to contain a reviewer's annotations without those annotations appearing in a printout, or could allow different versions of the document to be produced for different situations. To achieve this we have suggested a simple variation on the style table mechanism described above. This facility is obviously needed for hypertext because most of a hypertext is not presented at all until selected by the user.

3.3. Extensions and Interactive Documents

This section shows how the proposed extensions can be applied to screen based documents and hypertext in general and then looks in more detail at how they can be applied to two particular hypertext systems.

ODA allows a measure of flexibility in the layout and presentation of documents, but different views are not a substitute for proper interactive facilities. The basic problem is that the ODA layout process is sequential and page based — and several attributes reflect this. Any form of online editing requires extensions to the layout process to make it incremental and to allow the user to scroll around the document, but some more ambitious features desirable for screen-based documents are

(i) *An outline facility* — to display selected (usually high level) items, such as chapter and section headings, and ignore other items.

(ii) *Pop-up displays* — to allow the temporary display of additional information on demand. These can be used for the equivalent of footnotes, marginal notes, and glossary entries in paper documents.

(iii) *Folding* — to allow sections of a document to be hidden behind a 'button' on the screen and revealed on request. Folding should be allowed to any level, so hidden sections can contain further buttons.

(iv) *A linkage facility* — to enable users to follow links or cross-references automatically.

Item (i) is dealt with by style tables that select objects by class and required level.

Item (ii) is dealt with by changing to another style table to produce a pop up display and then changing back again when the display is no longer required.

Item (iii) is an extension of item (ii). The layout process needs be able to display *either* the button *or* the item(s) folded behind the button. One way to do this is to have both the button text and the folded components as subordinates of the *button object*. The button is closed when a style table is applied that displays just the button text, and it is opened by applying another style table that displays the folded items (and possibly the button text as well).

Item (iv) could be done in several ways depending on the type of link. Three possibilities are

- Move the current point of display to the target object.
- Display the target object (or subtree) as a temporary pop-up item.
- Include the target object (or subtree) at this point in the document.

These can be achieved with a combination of style tables and links. The style table specifies whether on not to display the linked object. When the style table is changed the linked object can be displayed as a new layout object (like a card), as a pop up item, or inserted inline with the surrounding content.

3.3.1. Modelling Guide Buttons in ODA

Guide [9, 10] is a hypertext system that supports a hierarchical model of a document and also allows cross-linking of information. A typical Guide document presents the reader with a summary consisting mainly of buttons. These can then be selected to reveal greater levels of detail as required. Buttons may be nested many levels deep. The reader selects only the buttons he is interested in, and if he finds he is not interested in the information revealed he can 'undo' the selection and fold the information back behind the button again. Guide is also a WYSIWYG editor. It allows the reader to edit the contents of the document and to add or delete buttons, thus becoming an author as well. The emphasis is on allowing the reader to tailor the document to his own requirements.

The overall Guide model is similar to ODA's hierarchical model, but with the added concepts of

(i) Folding logical items behind buttons.

(ii) Allowing more than one button to access the same logical items.

Guide's layout model is of a single long scrollable frame holding all content except temporary pop-up items. Using an ODA framework could enrich the Guide layout model. To show how the Guide model fits with ODA, we shall introduce two different types of Guide button and explain how they might be represented. (The examples use the UNIX version of Guide, which is similar to the version marketed by OWL for the Apple Macintosh [11] but differs in some details.)

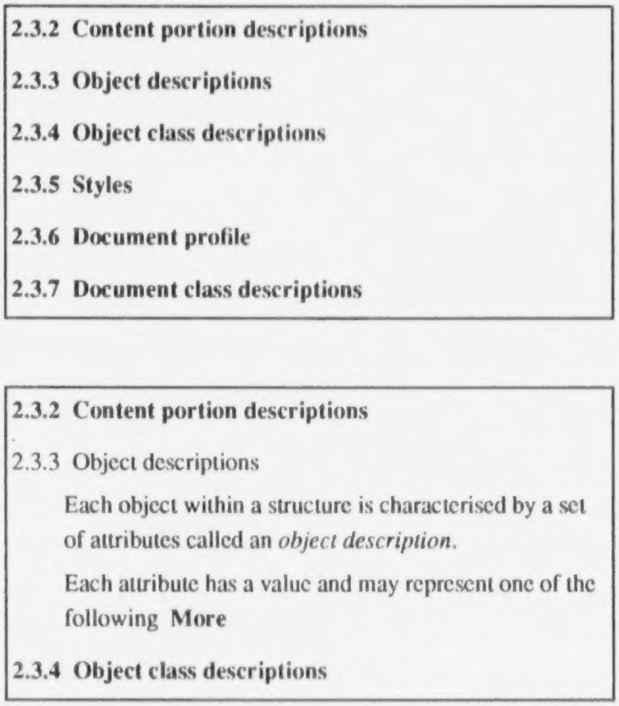
The commonest type of button is the *replacement-button.* When a replacement-button is selected, the button itself disappears and is replaced by information that may in turn contain further buttons. The replacement is inline, so surrounding text may be reformatted or scrolled out of the way to make room for the replacement.

Figure 4 shows two different views of a Guide version of part of the ODA standard. In Figure 4(a) the visible text is made up entirely of buttons giving section headings. (By convention, Guide buttons appear in a distinctive font — typically in bold — so that readers can recognise them.) Figure 4(b) shows the result of selecting the 'Object Descriptions' button. Two further buttons are shown within the replacement. The 'More' button is another replacement-button for the user to select if he requires more detail. The words in italics are a different type of button known as a *glossary-button.* If the reader selects a glossary-button an explanation of the term appears temporarily in a separate window.

To represent Guide buttons in an ODA document we would not set about defining a special new ODA object class for each type of button. Instead, for replacement-buttons, we would look first at the existing objects in a document class, decide which were appropriate as buttons, and apply style tables that would make them behave like buttons. Sections might be considered suitable for use as buttons, in which case the subtitle might be displayed as the button text, and the whole object displayed when the button is selected. Other classes of object (list items for example) might be modified for use as buttons by adding some abbreviated version as a button text component.

There are several variations on the basic replacement-button. The simplest form is the *local-button* where the replacement applies only to the button itself. This is the default type described above. Two other forms are the *definition-button* and *usage-button.* For definition-buttons the replacement applies not only to the button itself but also to usage-buttons with the same 'name'. (Guide provides a mechanism for attaching names to the buttons.) It might be more efficient to mirror this in ODA by providing usage-buttons with button text and a link to the appropriate definition-button object. This then becomes a general mechanism for attaching the subtree containing the replacement content to several places in the document.

Glossary-buttons are like footnotes, annotations, glossary entries, or other embellishments to the main document. Unlike replacement-buttons their replacement is not part of the main document, instead it is typically a short piece of pop-up text. We could represent glossary-buttons in ODA by defining a new 'Glossary-button' generic object with a *generator for subordinates* specifying a button text item and a 'Glossary-text' item.



2.3.2 Content portion descriptions

2.3.3 Object descriptions

2.3.4 Object class descriptions

2.3.5 Styles

2.3.6 Document profile

2.3.7 Document class descriptions

(a) Summary containing unexpanded buttons only

2.3.2 Content portion descriptions

2.3.3 Object descriptions

Each object within a structure is characterised by a set of attributes called an *object description*.

Each attribute has a value and may represent one of the following **More**

2.3.4 Object class descriptions

(b) Result of selecting 'Object Descriptions' button

Figure 4: Guide document showing (a) button and (b) expanded button

'Glossary-text' would normally be defined as a simple leaf object with character content (to represent the explanation text). However glossary-buttons are intended to provide the same explanation for each reference to a term or item throughout the document, so it is attractive to think of a variation, similar to the usage-button, with a link to the appropriate explanation text.

3.3.2. Modelling KMS Frames in ODA

KMS [3] supports a data model based on workspaces known as frames. Frames may contain text, graphics and image items, and individual items within frames can be linked to other frames. There is no built-in notion of hierarchical organisation and no concept of a linear ordering of information. Information is divided into frame-sized chunks and one chunk is displayed in each window on the screen. The reader follows links to view different frames.

In spite of this very general model, strong conventions have evolved for the format of frames and for distinguishing between hierarchical links and other links. Figure 5 shows the overall layout of a conventional KMS frame. (To avoid confusion this section will use 'KMS frame' and 'ODA frame' to distinguish the different meanings.)

The generic logical objects defined to support a standard KMS database would correspond to the KMS frame and the items within the KMS frame. Figure 6 shows the top levels of a possible generic logical structure.

The generic layout structure for a KMS frame would correspond to an ODA page with ODA frames representing the areas shown within the KMS frame in Figure 5. *Layout object class* would be used to direct each KMS frame into a single instance of this ODA page, and *layout category* and *permitted categories* would be used to direct the different logical items into the appropriate ODA frames.

The 'tree' and 'link' items would be set up like the replacement-buttons described for Guide in the previous section. Thus 'tree' items would be like definition-buttons and would have two subordinates: the button text to be shown in their parent KMS frame and another KMS frame (to be shown if the button is selected).

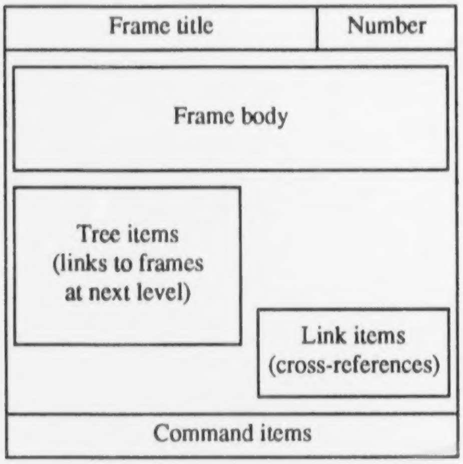


Figure 5: Layout of a typical KMS frame

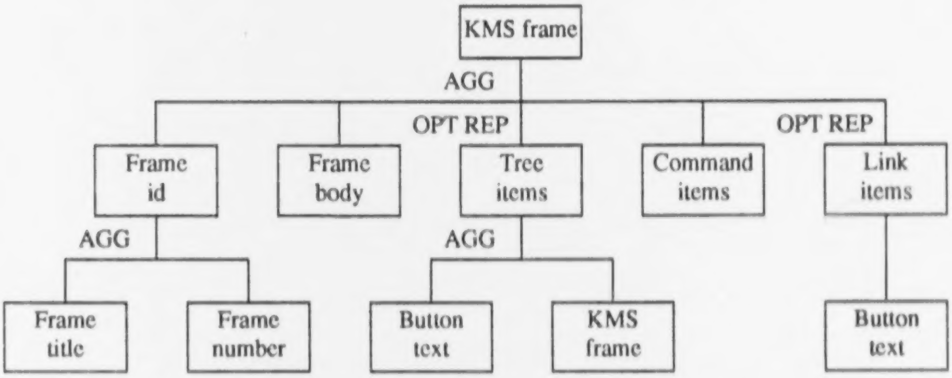


Figure 6: Generic logical structure for a KMS frame

The 'link' items would be similar to usage-buttons. They would have button contents to be shown in their parent KMS frame, and a link to the remote KMS frame. The layout process could be relatively simple as it only needs to display complete KMS frames and to follow the primary and secondary links to further KMS frames given in the 'tree' and 'link' objects.

4. Conclusion

A great deal of effort has gone into the production of the ODA standard and much practical experience has been gained. A new hypertext standard should not try to reinvent the wheel. We believe the best solution is to combine the existing expertise enshrined in the ODA (and SGML) communities with the expertise in the hypertext community. We must avoid having two or three separate standards and squandering the efforts of the few experts available.

Acknowledgements

We would like to thank British Telecom and the SERC for their support of research projects on document structures and ODA.

References

[1] *Information Processing - Text and Office Systems - Office Document Architecture (ODA) and Interchange Format* ISO 8613-1988, International Org. for Standardisation, 1988.

[2] L.Lamport, *LaTeX user's guide and reference manual*, Addison-Wesley Publishing Company, 1986.

[3] R.M.Akscyn, D.L.McCracken and E.A.Yoder, 'KMS: A Distributed Hypermedia System for Managing Knowledge in Organisations' *CACM*, **vol. 31 no. 7,** pages 820 - 835, 1988.

[4] J Conklin, 'Hypertext:introduction and survey' *IEEE Computer* **vol 20, 9,** pages 17-41, 1987.

[5] P.J.Brown, 'Do we need maps to navigate around hypertext documents?' *Electronic Publishing — origination, dissemination and display*, **vol 2, no. 2,** pages 91 – 100, 1989.

[6] *Information Processing - Text and Office Systems - Standard Generalised Markup Language (SGML) ISO 8879-1986*, International Org. for Standardisation, 1986.

[7] *Information Processing - Text Composition - Document Style Semantics and Specification Language ISO/IEC DP 10179*, International Org. for Standardisation, 1989.

[8] F.C.Cole and H.Brown, 'ODA modifications/extensions version 3', submitted to *ISO/IEC JTC1/SC 18/SWG*, January 1990.

[9] P. J. Brown, 'Interactive Documentation', in *Software — Practice and Experience*, **Vol. 16, No. 3,** pp 291–299, 1986.

[10] P. J. Brown, 'A Simple Mechanism for the Authorship of Dynamic Documents', in *Text Processing and Document Manipulation*, ed J. C. van Vliet, pp 34–42, Cambridge University Press, 1986.

[11] *Guide: Hypertext for the Macintosh*, OWL International Inc., 1986.

Standards for a Hypermedia Database: Diachronic vs. Synchronic Concerns

Gregory Crane
Perseus Project
Department of the Classics
Boylston 319
Harvard University
Cambridge MA 02138

This paper outlines the perspectives of a professor in one traditional branch of the humanities (Classics). My colleagues and I are engaged in creating a hypermedia database on ancient Greek civilization, but our work is intended to explore the generic issues of building a complex hypermedia database, and Perseus was conceived as a model for what should (and no doubt should not) be done. We have encountered a number of problems along the way that must be solved before information disseminated in a hypermedia environment can have more than marginal impact on intellectual activity. This paper addresses hypermedia databases: although much of our work revolves around texts and still images, we can see that sound, animation, and motion video are also basic categories of information. This paper at least views hypertext as a subset of hypermedia.

The argument of this paper can be summarized simply. Standards for hypermedia must emerge before hypermedia databases can be fully useful, but long-lived standards can only emerge after we know much more about how people will use hypermedia databases. Since we can do qualitatively different things in a hypermedia environment, we must assume that usage patterns will emerge. Practically speaking, we can expect to see short term interchange tools so that we can move data from one hypertext system to another, but we should be prepared to abandon these standards if they prove too inflexible. The rest of this paper outlines some pragmatic concerns.

Standards can be viewed as working in two dimensions, synchronic and diachronic. Synchronically, hypermedia standards would allow all hypermedia systems at any one time to exchange and share information: thus, *NoteCards*, *HyperCard*, *Intermedia*, *HyperTies*, *Guide* etc. could all exchange the same data. Synchronic standards are, in some measure, feasible, and are a crucial first step. This paper, however, focuses on diachronic continuity: the same hypermedia database must be equally useable now and for many years to come. In fact, any hypermedia database that fits cleanly into any existing hypermedia system will probably not long survive. Synchronic standards will provide us with experience and knowledge that we can use to create truly diachronic standards. If we are lucky, synchronic will evolve into diachronic, without sharp breaks in continuity.

For many, synchronic is more important than diachronic continuity. We do not need to preserve for centuries all the product documentation for every computer system available in 1990. Even a 1970 paper on new directions in punch card technology, for example, would have little appeal to the engineer today. The Historian of Science may some day wish to study this technology, but we cannot preserve everything. In such areas, information must be disposable.

The notion of disposable information has profound implications. If one's ideas will only be valuable for five or ten years anyway, then the author may not care very much if those ideas are stored in a hypermedia system that is itself equally ephemeral. *Fress*, an early hypertext system released at Brown in 1971, was demonstrated at Hypertext '89, but it appeared there as an historical artifact rather than a living system (its official title was "A Blast from the Past: The Last (?) FRESS Demo". For others with a potential interest in hypermedia such as textbook publishers, short-lived systems are ideal, since they can thus attack the used-textbook market and force students to buy new electronic "textbooks" with greater regularity.

It is hard to emphasize how destructive such attitudes are. True publication, however, implies that a document will be part of the public record for an indefinite period of time, not just for a few years. In many disciplines no scholar can afford to lavish time on creating documents that will not last at least thirty years and, hopefully, much longer. This holds true not just for humanists creating tools such as critical editions of authors (e.g., Homer, Chaucer), dictionaries and commentaries, but for many other areas as well. Anthropologists, for example, working in Central Africa or Latin America have their own questions in mind, and their own conclusions may soon become dated. But they also create ethnographic descriptions of societies that are rapidly changing. Their published ethnographies may be our best (even our only) records of those societies, and these must become permanent part of our information infrastructure. We are constantly adding to our basic *record* of the world, and this record must be maintained for an indefinite future.

The author who creates information and the system that stores that information are only two aspects to a larger whole. Consider, for a moment, one other critical group that must also embrace the idea of hypermedia and for whom longevity is even more important. The librarian must be able to leave information "on the shelf" for centuries rather than decades. No document will last long if it is not preserved as a regular part of our research library system. I would like to emphasize that a standard that does not meet the most stringent needs of research librarians is, at best, a crude stopgap and, at worst, quicksand that will trap and overwhelm the unwary, and that will make subsequent travellers view hypermedia with distrust.

The problem from our perspective may be summarized as follows. Hypermedia systems offer tremendous potential and may ultimately revolutionize the way in which research is performed and disseminated. Hypermedia cannot, however, have the impact that it warrants until we can provide diachronic continuity. A database that runs on ten systems now (and thus provides synchronic continuity) and zero systems a decade from now does scholar and librarian little good.

Problem 1: Exchanging Data

Exchange standards offer one obvious approach to the problem of diachronic continuity. If we can exchange database *Fred* between N different systems at any one given time, then there is a high probability that *Fred* will be able to move into new systems that have not yet appeared. *Fred* may not take advantage of all the capabilities of its new environment just as a black and white silent movie does not exploit the full capabilities of the television on which it may be viewed, and in some ways performance in the new system may be weaker (e.g., video has inherently less resolution than any film and thus cannot reproduce all the information in any one frame of the film). But at least *Fred*, like the silent movie, will still be accessible.

Converting hypermedia databases from one system to another is much more complex than transferring silent film to video, more complex, perhaps, than the problem of converting a play into a movie. For while the play and the movie have profoundly different options open to them, the script of the play (in most cases) provides a common linear path which both can share, and a movie can imitate the conventions of the stage.

The conversion from one hypertext system to another may well prove more analogous to the problem of machine translation. Existing hypermedia databases and even standards for particular types of information (such as the SGML standard for text) are generally closer to syntax than semantics. They illustrate how various objects are put together, but they can only incorporate a limited amount of information about why the objects are put together in that particular way. The designers of the hypermedia database will unconsciously tend to rely on the peculiarities of the system that they are using. Authors organize their data differently when using a system in which scrolling windows can contain large documents (e.g. *Intermedia, Notecards*) than when working with an inherently "chunky" hypertext system (one built around many small cards)

Consider two examples:

1) *HyperCard* can easily store a hierarchical map. The user begins with a view of the world, zooms into a view of a particular country, and then calls up the plan of a particular city. A user can implement such a map easily with buttons containing `goto`'s, but will an

interchange program be able to recognize that these buttons represent, in fact, a logical hierarchy? If the interchange program cannot make such inferences, will it produce results like the machine translation system that interprets "time flies like an arrow" as "time-flies enjoy arrows" or as "time the flies (i.e.with a stopwatch)". If hierarchical structures of one kind or another are to be a building block for hypermedia systems, then all such systems must contain primitives that recognize these structures.

2) Much discussion has gone into the creation of links between anchors in various documents. Document X would have a link to an anchor in Document Y, and the anchor would identify a particular point or selection in Document Y. This is a critical and generic concept, but, in some contexts, it replicates a function that text strings implicitly perform: e.g. "Shakespeare Macbeth 1.7.1-2 'If it were done . . . . quickly'" defines a precise subset of the text. The text string is a high level construct that does not depend upon anchors into one particular document: it will work equally well whether the Riverside Shakespeare or the Folger edition of Macbeth is online. Does an automatic linking protocol really constitute an advance over such a reference, or even over a standard journal reference (e.g. "*HSCP* 91 (1987) 175 note 60")? If document (or an object in a museum for that matter) does not already have an anchor of this kind, then that information has not been published in any meaningful sense. Publication presupposes the existence of canonical citation schemes. Where canonical citations schemes do not exist or are imperfect, then information, like a misshelved book, is lost.

Second, publication (as in *Augment*) cannot be retracted. A statement, once it has been placed in the public domain can never be changed: it can be commented on, and its author may recant, but the statement must remain a part of the record. A publication system (as opposed to an authoring system) should not accept vanishing links.

New products such as *SuperCard* and *Plus* do attempt to interpret all the information within a *HyperCard* stack, but only because their own model of the world is a superset of the *HyperCard* model. Once a document is truly converted to either *SuperCard* or *Plus*: i.e., once it takes advantage of elements in the *SuperCard* or *Plus* model that are not available in *HyperCard*) then it cannot easily move back to *HyperCard* or even laterally to from *SuperCard* to *Plus* or vice versa. As soon as hypermedia systems begin to change their view of the world, then different systems will have different abilities. Translating from one environment to another becomes an interpretive act, in which human intelligence may prove irreplaceable for the forseeable future.

The rest of this paper will cover problems that we in the Perseus Project have encountered in building a hypermedia database on ancient Greek civilization. The domain is relatively compact: 40 and 100 megabytes of source texts in original Greek and English translation, a dictionary, a small encyclopedia, essays, maps, plans, and 5,000 to 10,000 images of Greek sites, monuments,and art objects will provide a solid foundation for the study of this subject. Nevertheless, the problems inherent in managing such a heterogeneous database of this magnitude are substantial.

More importantly, this data is intended to serve a wide audience. First, it aims at different levels of expertise: the undergraduate in a general course and the professor doing research. Second, it aims at various kinds of expertise: the same data should be useable for the study of literature, art, history, linguistics and other subjects. In fact, both distinctions are related: the more accessible information about art is, for example, to the freshman, the easier it can be for literary critics, who do not now have easy access to that information, to use it in their work.

Our work is, to a large extent, an experiment within which we are trying to identify the basic data structures with which people work. Objects such as dictionaries, atlases and museum catalogue entries have evolved certain fairly stable forms that are based on functions that people seek to perform. As these tools migrate into an electronic environment they can perform new functions and their forms will inevitably change. Until we have a better idea of what these new functions will be, however, we are not in a good position to build environments in which the form of information can evolve.

Data Models and Approaches: Some Concrete Problems

Every discipline probably has its own proprietary data models which every expert must internalize. Thus, the mathematician must know how to create and present a logical proof, while the chemist needs to provide certain kinds of information when describing an experiment. The student of ancient Greek literature knows how to read and to use a scholarly edition of a Greek text, while the archaeologist knows how to work with objects discovered on a dig. Hypermedia standards must provide a model in which each group can express as many significant features as possible. They must at least replicate the functionality of printed texts, but should also allow people to perform new operations.

Defining a data structure is not an easy task. Even if we have a model that satisfies one group, another group may want to use the same information in different ways. The following section provides two general examples of the iterative process that we have had to undergo. The examples are fairly specific but they illustrate how difficult it will be to define what some people have in mind when they think about such basic categories as archaeological objects and source texts. The problems below are very specific, and domain experts in various fields will have to create the actual specifications for these data structures. Nevertheless, the standards that evolve for hypermedia databases will determine how feasible it is for the domain experts to organize their information. The more effectively authors can organize their data, the more useful the underlying standards will prove. Particular and domain specific as these problems may seem, they address fundamental data types. Until hypermedia standards provide a platform that supports such data types, hypermedia cannot play a major role in the publication or the long term archiving of information.

The classicist discussing Greek religion may, for example, use the painting on a Greek vase as evidence. He may point out that there is a man is leading a bull to an altar, that the man holds in his hand a sacrificial cake and some barley to sprinkle over the victim. He may draw attention to the kind of knife held or some other particular of the scene. In this context, a single one bit deep bitmap may well contain all the information necessary, and the expert in Greek religion might want to collect a large number of such images.

The art historian might want to study the style of the painter who created the picture. He would need to study very subtle details (such as the way in which anatomical details such as eyes or knees were rendered), but such detail will almost certainly lacking in the bitmap. The classicist can build up an enormous database of images which then prove to be of little use to his or her colleagues in archaeology or art history.

Worse, the art historian may actually conclude that one bit deep images are all that the computer can offer and thus turn away from the new medium. Likewise, many videodiscs (to choose one technology) simply imitate image libraries, even though a single video image cannot approach the clarity of a 35 mm slide. The art historian may thus conclude that a videodisc s just a poor substitute for a slide archive, but if the videodisc designer takes advantage of the storage space, then he or she can store multiple views of each complex slide and can provide much more information. A videodisc that stores details of every head in a series of paintings contains information that the slides do not, for the ability to move directly from head to head to head allows the reader to see the images in a different way than would the undifferentiated slides. In the case of images, the media available to us so far have been so primitive, that few of the scholars who really care about art, for example, have been able to see much promise in electronic databases at all.

Suppose, then, one builds up a database that serves the needs of both the classicist and the art historian. Thus, when we in the Perseus Project, for example, commission new photography of an art object, we collect multiple views: dozens for a single vase with many figures. A videodisc thus will have enough color views so that it will allow scholars to see more detail of the objects on the disc than could any affordabe printed publication.

The case is not, however, closed. Up come the anthropologists, also expert in handling physical remains. For them, the detailed views are extremely useful, but they want to reconstruct day to day life of the period. The database of images focuses primarily on the most elegantly painted and attractive vases: the art historian wants to study the aesthetics of classical Greece; since carefully drawn and visually harmonious vases contain much of the information that the general classicist needs, the two groups work well together. The anthropologist wants to see what people actually used, not just the most polished specimens, but the coarse, hurriedly drawn pieces as well. Perhaps, he does not even want vases in particular, but tools and other objects that illustrate the kind of work that people performed. Again, the invidual entries for each object may be quite attractive, but the anthropologist might argue that the collection as a whole provides a biased picture of the ancient world. Nor are the anthropologist's complaints necessarily limited to gross selection of objects: he or she have very different kinds of questions that they are going to ask and if a database is going to serve their interests, then its structure will undoubtedly need to be changed.

Literary texts offer similar problems, for different groups view texts in different ways. The text of *Moby Dick*, for example, is conceived of as a fairly stable text stream. The critic will refer to a particular chapter or perhaps a page in a particular edition, but what Melville wrote is clear enough. It is relatively easy to build a publication model for "text" if we think in terms of nineteenth century English and American novels (and if we do not think too deeply about the problem).

CHECKED UP TO HERE.

If we apply this concept to a text that was transmitted in manuscript, this model is inadequate. Every time a large document is copied by hand, mistakes appear, and these mistakes become compounded with each new copy. Over the course of centuries, many variant forms of the text evolve and only with the printing press can this process of dissolution be arrested. Nevertheless, the damage is done: editors must choose between many competing variants, and must tell the reader when they choose a reading from manuscript X or Y. The reader needs, at a minimum, to see what variants are available for any passage of text. Ideally, the system should be able to show the reader where editor A chooses different readings from editor B, or to show, for example, which corrections in the text were suggested before 1800.

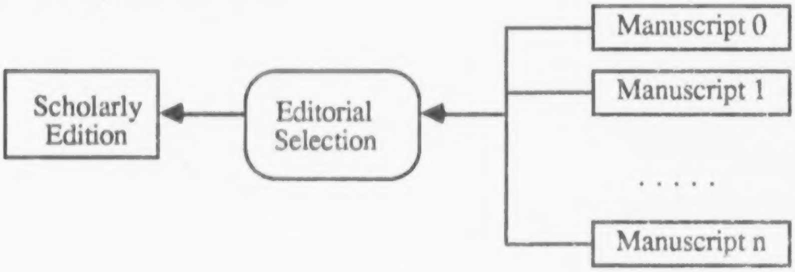


Figure 1: Simplified view of a scholarly edition derived from various "manuscripts". Every line of text may involve an "editorial selection."

Again, addressing both the nineteenth century novel and ancient Greek literature forces us to broaden our model of what a text is. Nevertheless, we are not finished. Consider a popular text that appears in various forms over a number of centuries. In the case of the Greek poet Aeschylus, for example, we assume that there is an original source text (i.e., what Aeschylus actually wrote) that we are trying to reconstruct. Ideally, we *could* treat Aeschylus like Melville if we had an authoritative edition of Aeschylus. In the case of a popular story, we may have multiple versions, none of which is associated with any dominant owner and each of which is essentially just as important as the others. Each version of the story may itself have its own manuscript tradition, but now we must consider a kind of compound versioning: a story consisting of multiple versions each of which has numerous textual variants.

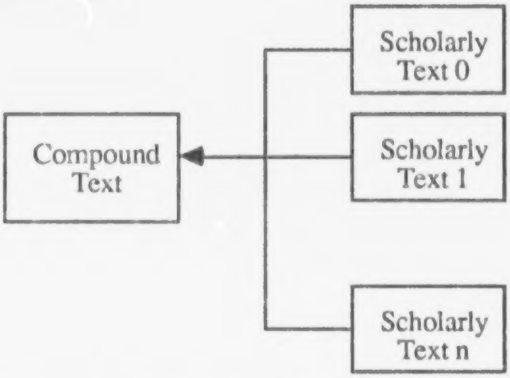


Figure 2: A compound text, consisting of n scholarly texts (each of which may be constructed from a variety of manuscripts).

On the other end, even the category of "manuscript" is not completely simple. A document may be preserved on a stone or clay tablet. The writing system used to store this text may be crude, and scholars may need to provide normalized transliterations that follow conventional spelling rules or add some standard kind of information (thus many editors of Greek inscriptions add accents to their final editions). In such cases, an edition may include (1) a picture of the inscription, (2) a transliteration of the inscription without accents or word breaks that simply, (3) a regularlized form. The physical medium may be stone or (as in the case of much Akkadian and Sumerian material) clay tablet, but in many ways the problem is similar to that faced by someone transcribing a sound recording made by the speaker of a little known language. The ethnographer may well want to include a narrow phonemic transliteration. Thus, we might outline the structure of a source document (of which a "manuscript" is one example) as:

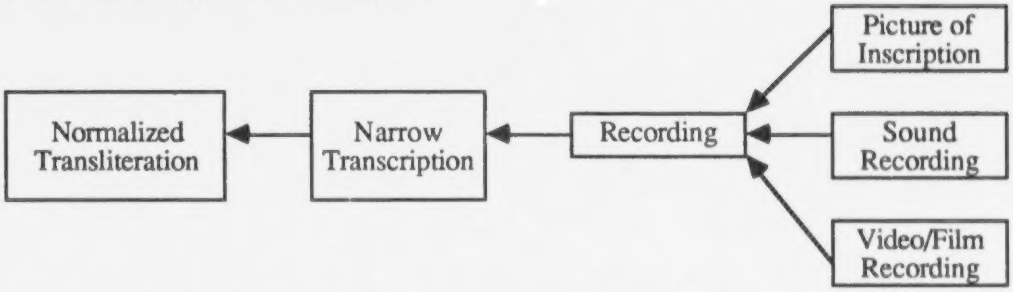


Figure 3: Diagram for one taxonomy of source documents (such as a manuscript or inscription).

This diagram presents a basic data model that will solve many of the problems for storing nineteenth century novels, Greek plays, Akkadian myths, Greek and Akkadian inscriptions, and an anthropologist's verbal recordings made in the field.

The particulars of this simplified model are less important than the process that led to its creation: had we standardized around the nineteenth century novel, the Greek play or the inscription, we would have adopted an impoverished data model. We need to view in as much detail as possible as many different kinds of text as we can before we assume that we know what a text is or what it can do. A system that can handle these functions must address links not simply from one document to another, but between text, pictures, sound and motion video. Until we have systems that actually perform these tasks, we will not be sure that our standards actually account for the problems that people need to solve. This kind of analysis has barely begun, and we have a long way to go before we reach any consensus as to how any basic categories of information should be organized.

Hybrid Data models

So far we have talked about simple data types that have analogues in the world of print. We can insulate the individual components of data from the vagaries of any one system by storing information in the most powerful medium possible. Thus, we at Perseus have pragmatically chosen to expend extra effort so that our information will be useful for a longer period of time: drawings are stored not as bitmaps but in Postscript; for still images we use 35 mm film rather than video. A single Postscript can generate multiple bitmaps at varying resolutions, and whatever the future of Postscript itself, subsequent graphic formats will probably be able to absorb most of the existing Postscript data. We will thus be able to upgrade our site plans and drawings to systems that do not rely on bitmaps. Slides, though not electronic, contain far more information than we can now reasonably store in digital form. Should new formats such as HDTV actually arrive within the next five to ten years, film will convert much more elegantly than inherently crude NTSC video signals with their limited resolution. None of the hypermedia or hypertext systems currently available can recognize sophisticated text structures that one can create in format such as SGML, but we store our texts in SGML and will be able to take advantage of more powerful hypertext systems as these emerge.

Efforts are already underway to provide workable standards in at least some of these individual areas. The Text Encoding Initiative, funded primarily by the NEH and EEC,[1] is a widely supported effort to build basic document formats for humanists within the framework of SGML. Storing images as slides or as postscript drawings is a pragmatic hedge rather than a workable standard.

Work on texts or images in isolation is only part of the problem, for these are only some of the basic components out of which a hypermedia documents might be constructed. Once we know how to handle these individual pieces, a hypermedia system must then be able to make the individual pieces work together as a whole. If an historical source text, an atlas and a database of topographical images (i.e., pictures showing buildings and places) all exist in the same database, then it can become much easier for the person going through the historical document to locate places on a map and even to call up images of what that place looks like now. Someone, for example, reading in the Greek historian Herodotus about how the Greeks defeated the Persians in the battle of Salamis might thus call up a map on which Salamis appears, then view color images of the strait in which the battle was fought or the hilltop from which Xerxes, the Persian emperor, viewed the battle.

Once traditionally discrete bodies of knowledge such as text, atlas and image archive, can dynamically interact with one another, then new compound document types become feasible. A narrative on the battle of Salamis might consist of (1) links to the relevant text sources, (2) a map of Salamis with various buttons which were in turn (3) links into the image archive showing what the strait of Salamis or the hilltop of Xerxes looks like. Nor should such links be entirely passive: an animated version of the battle could be overlayed onto the generic map. Rather than calling up an entire picture, the system should be able to crop a particular detail, so that the view frames that particular hill, for example, on which Xerxes may have sat. A document may dynamically abstract and shape data from a larger data base.

Such interactive and dynamic links fulfill logical needs and will inevitably become part of the author's repertoire. An author should be able to create a document that pulls together and performs operations on material in a larger database. It is not enough, however, to be able to perform such actions in a particular system in a particular time. Once an author has published such a hypermedia document (perhaps as part of a book interpreting the wars between the Greeks and Persians), then scholars a century later must be able to view that hypermedia document and see exactly what the author saw. If this diachronic continuity is not feasible, then the hypermedia document may have been distributed but cannot properly be said to have been "published". True publication implies that the material will remain available for the indefinite future.

---

[1] The Project Director for this is Dr. C. Michael Sperberg-McQueen, of the University of Illinois at Chicago Circle.

Conclusions

We should move as quickly as we can towards some kind of synchronic interchange standard for hypermedia. We need to learn how well we can move fairly complex sets of data *and functionality* between diverse systems (e.g. *HyperCard, Intermedia, Notecards*). Once we are able to perform this task for some data, we may well decide that the interchange format that developed is, in fact, too inflexible. With luck, this interchange format will be a powerful platform that can evolve into a standard that will provide scholars and archivists with the diachronic continuity that they require. We must, however, be prepared to discard that format.

The risk is probably greatest for those of us creating databases: until we have diachronic standards, the information that we create may be available in libraries, but it will not be part of the library system. It will be distributed, but not truly "published." Nevertheless, we cannot make much progress on standards without applying them to substantial and fairly complex bodies of data.

From a practical point of view, we suggest that those developing interchange standards should plan to work from the beginning with one or more databases at least as large and complex as that of the Perseus Project. An interchange system that can move this database back and forth between three or more different hypermedia systems may not be perfect, but an interchange system that cannot satisfy this practical requirement will certainly not support the much greater challenges that it will face.

The Trellis Hypertext Reference Model

Richard Furuta* and P. David Stotts
Department of Computer Science
University of Maryland
College Park, MD 20742

Abstract

We describe a hypertext "meta-model"—one that provides an organization for the architecture of a hypertext model. The specific meta-model presented was developed in the context of the Trellis hypertext model. However the organization seems generally applicable to other models as well. As such the meta-model may be a good candidate for a hypertext reference model, and so we call it the *Trellis hypertext reference model*. In this report we first describe the Trellis hypertext reference model, and then discuss the relationship of some hypertext-defined concepts to the reference model.

1 Introduction

As a side-product of our work developing the Trellis model of hypertext [SF89a], we have defined a "meta-model" that provides an organization for the architecture of the hypertext model. It is the purpose of this report to describe this meta-model within the context of the Trellis model and further to suggest that it is applicable to other models of hypertext as well. As such it may serve as an appropriate framework for the development of a general hypertext reference model. In this report we shall call the "meta-model" the *Trellis hypertext reference model*, abbreviated as *r-model*, as a reflection of this application. The model of hypertext itself will be called the *hypertext model*, or more simply the *model* throughout the report.

The Trellis hypertext reference model is based around a collection of representations of the hypertext at different levels of abstraction. Abstractions range from the hypertext as a collection of abstractly-defined independent components through more concrete representations in which the characteristics of the hypertext's physical display have been established, to the view of the hypertext that is projected on a physical display device for the benefit of the person reading the hypertext. The representations at a particular level of abstraction depend upon representations at a greater level of abstraction, and these dependencies are shown within the r-model.

A description of the r-model follows in the next section. Section 3 discusses how selected components of existing hypertext systems and models fit into (or are omitted from) the r-model.

---

*Supported in part by a grant from the National Science Foundation, CCR-8810312.

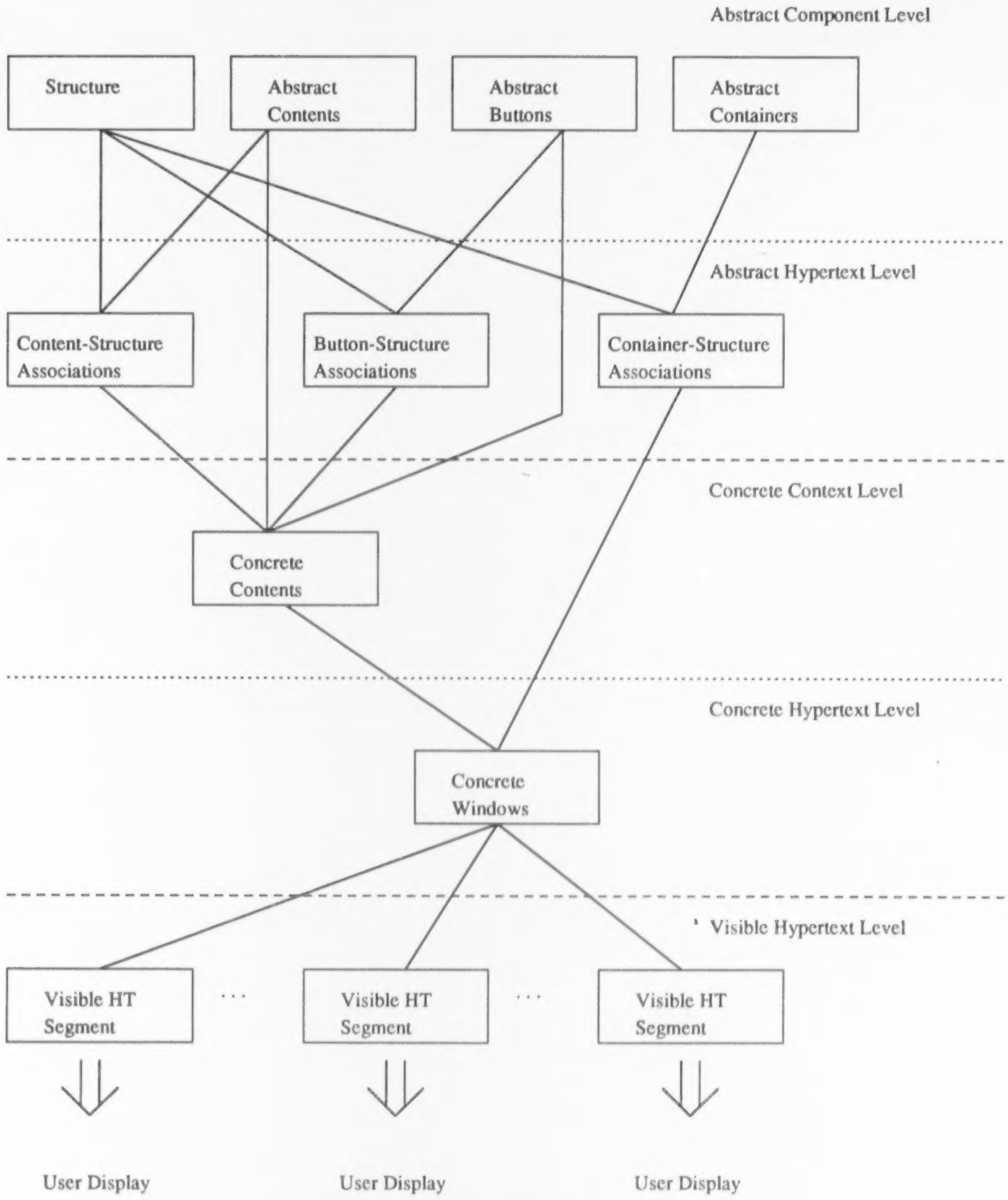


Figure 1: The Trellis Hypertext Reference Model (the r-model)

2 The r-model

The r-model, shown symbolically in Figure 1, is separated into five logical levels. Within each level is found one or more representations of part or of all of the hypertext. Speaking quite broadly, the levels may be grouped into three overall categories: abstract, concrete, and visible.[1] The abstract component and abstract hypertext levels define an abstract representation of the pieces of the hypertext and of the hypertext itself. These abstractions are transformed into more concrete representations of the hypertext in the concrete context and concrete hypertext levels, representing first the presentation of the hypertext's content and then the mapping of that content into the displayed windows. The resulting concrete windows are then viewed, producing one or more displays on one or more physical display devices. In summary, the representations in the abstract component level are at the greatest level of abstraction and those in the visible hypertext level are at the lowest level.

Each representation is shown in the figure as a box. A representation is itself an abstract concept—a consistent presentation of the hypertext elements of interest. Representations in the r-model may depend on the representations at a greater level of abstraction. Such a dependency is shown in the figure as an arc between the representations. Because a representation's dependencies are on those representations at a greater level of abstraction, and not on those at the same or lower levels of abstraction, the abstract and concrete levels in the diagram are further subdivided. It is worth emphasizing that a representation may not actually correspond to a separately-identifiable "physical" representation of the hypertext; for example, the representation may be expressed as a mapping between elements of more abstract representations.

We will now focus in turn on each of the levels of the r-model. In the following sections, we will describe the level, its representations, and discuss the dependencies on representations at higher levels.

2.1 Abstract hypertext

An abstract hypertext description specifies a hypertext and its components, but does not describe the details of how the hypertext is to be presented to its reader.

2.1.1 Abstract component level

The organization of the three highest levels reflects a separation of the hypertext into *structure*, *content*, and *context*. The structure represents the elements of the hypertext and their relationships. The specific content of the hypertext as presented to the system's user reflects the context within the structure in which the content appears—in other words, the display of the content is modified to reflect its context.

The representations within the abstract component level present the components that will be associated with one-another to form the hypertext. Within the context of this level, the representations are independent of each other—such associations will be made at lower levels of abstraction. Our abstract view of a hypertext separates out the hypertext's structure from the elements that many users perceive as composing the hypertext. In other words, the structure, perhaps a directed graph, is separated from the collection of contents that are to be displayed to the reader and the collection of "buttons" that will be selected by the reader when moving from location to location in the hypertext. Additionally, it may be the case that the view of the hypertext presented to the reader combines together independent content elements into an integrated whole. The presence (or absence) of such composition is also represented abstractly at this level. We will now consider each of the representations in turn.

---

[1] The choice of these levels of representation parallels and expands Shaw's model of printed documents [Sha80] which identifies abstract, concrete, and viewing mappings for the document.

One natural representation for the **structure** of the hypertext is as a network. In our own work, we use a Petri net structure, which provides automaton semantics as well as the network representation. However other graph-based structures are appropriate as well—for example automata such as deterministic finite automata or data structures such as directed graphs, trees, or lattices. The structure of the hypertext need not be limited to networks; indeed, it may be desirable to use representations that are not graph-based in form; for example constraint-based descriptions. Note that even in graph-based representations, there is no requirement that the elements of the structure be fully-connected. The necessary characteristics of the structure representation is that it provides the "placeholders" that will be associated with the hypertext's content and that it describes the relationships that exist among these placeholders.

The **abstract content** is arbitrary in form. It may, for example, include textual, graphical, animated, or perhaps even audio and video material. The content may be specified directly or may be the result of a computation. While it does not contain links, it may incorporate markers that define a collection of potential locations for the mappings of links and their presentations that occur in lower levels of the r-model. The content may be described in a form that is independent of the eventual characteristics of its display, or indeed it may be described in a form that is highly dependent on the eventual display. Because of the flexibility of the mapping from content to structure in the next level, however, a display-independent representation seems most appropriate.

The structure representation identifies the relationships among content elements but does not indicate how those relationships will be shown for selection by the hypertext's reader. The **abstract buttons** are abstractions of the ways in which the relationship can be displayed. Abstract buttons may themselves have content and an associated type. The content is provided to specify *what* will be shown when the button is displayed. The type is needed to specify *how* the button will be displayed and other characteristics of its behavior on display and selection. As with the content of the abstract content, the content of the abstract button is variable in form—in implementation it actually may be computed or it may be statically defined.

The final component in this level, the **abstract containers**, differs from the others in that it is an abstraction of how the pieces of the hypertext will be combined when shown to the reader (*how* it will be aggregated and combined for display), and not of *what* is in the hypertext. For example, if several content elements are displayable, one possible presentation would be to show each element separately while another would be to combine the separate elements into a composite, which would be presented to the reader as a unit. In the first case, one could say that a separate container had been associated with each separate content element, while in the second case, one container would hold all content elements. Such characteristics are abstracted at this level by the abstract containers.

2.1.2 Abstract hypertext level

The elements of the abstract component level are not connected together, as will be necessary to form a hypertext. This association is performed in the abstract hypertext level. The abstract hypertext level does not, however, describe *how* these associations will be presented within the display of the hypertext. This is left to the concrete context level.

The **content-structure associations** map together elements of the structure and elements of the abstract content. In a graph-based structure, one natural association is to map the content elements to the nodes of the graph. No restriction is expressed in the r-model on the kinds of mappings that are permissible—for example it may be useful to map a single content element to multiple locations in the structure, or conversely to map multiple content elements to a single location. In our own work, we have found the ability to map a single content element to multiple locations to be particularly useful. We have also found it useful to completely substitute a new collection of abstract contents and of content-structure associations while retaining the same structure—for example for related hypertext versions, where one may perhaps be a translation of the other.

The **button-structure associations** map the structure's relationship and abstract buttons. A natural association in a graph-based structure is to map the abstract buttons to arcs in the graph. In our Trellis hypertext model, based on Petri nets, the mapping is between the class of node called a transition and the abstract buttons (i.e., there is no mapping of arcs in this particular graph structure). Again we emphasize that there are no limitations expressed on the form of the mapping, although we have found a one-to-one mapping to be the most useful.

Finally, the **container-structure associations** describe the association of the structure, or of portions of the structure, to one or more abstract containers. One use of this association is to permit grouping of elements of the structure, which might in turn be displayed to the reader in a single physical window. Different kinds of composite displays would be represented as associations with different types of abstract containers. In general, the container-structure associations allow the partitioning of the subsequent display of the hypertext into one or more possibly overlapping pieces.

2.2 Concrete hypertext

Assume that a hypertext is presented to its reader or readers in one or more windows on one or more physical display devices.[2] A concrete hypertext description specifies what the contents of each of these windows will look like but does not tie down how the windows are to be arranged on the display. For example, one particular window may be shown on several separate displays. Furthermore, the characteristics of the displays may be different; in this case the subsequent viewing description will also indicate how the different visible effects specified by the concrete description are to be rendered on the displays.

2.2.1 Concrete context level

The previously-described levels have defined an abstract hypertext in which the content and the buttons have been associated with the structure. However, the abstract hypertext description does not indicate how links are to be presented in the display of the content. Such considerations of the mapping from the hypertext's abstract representation to its physical representation are addressed in the concrete context level.

The concrete content presents a physically-oriented description of the hypertext. This mapping must address the following points:

---

[2] Here a window contains a concrete view of the hypertext (or portion of the hypertext) to be presented to the reader.

- How is the abstract content to be formatted to fit within the display region?
- How are the buttons to be displayed? Will the display of the button modify the display of the content or will the buttons and content be displayed independently? For example, in our initial Trellis prototype (αTrellis), we have provided externally represented buttons. In our subsequent prototype (χTrellis), we have also developed means for specifying that the button is to be represented as a highlighted string within textual context [FS89a]. Note that button displays are not necessarily static; in some cases the display of the button depends on computed material (which itself may depend on the structural relationships in the hypertext). The button represents the source of a link in the hypertext.[3]
- Is the target of a link associated with a content element as a whole, or is it associated with a particular location within that content? Does the display of the target affect the display of the content?

The mappings on this level do not rely directly on the structure (abstract component level) because the structural relationships have been "encoded" into the representations of the abstract hypertext level.

2.2.2 Concrete hypertext level

The concrete context level has defined a set of concrete content elements in which a concrete representation of the content has been merged with concrete representations of the buttons. The concrete hypertext level maps those concrete representations into a set of windows for display. The mapping, which produces the **concrete windows** representation, also requires that link-based interrelationships among the windows be determined. For example, the process of following a link can result in several different display mappings: the display of the target of the link could replace the display of the source, could be shown in addition to the source, or could modify the display of the source, with both being shown in the same window.

When the concrete windows representation has been formed, the presentation of the hypertext has been determined but the details of how and where the windows are to be displayed has not. For example, multiple windows may be shown to a single reader on a display or a particular window may be shown to several reader simultaneously on separate displays. Indeed, a particular reader may have several physical displays at his disposal, and different displays may have equivalent but different means for achieving particular visual effects. Such considerations are addressed in the next level.

2.3 Displayed (visible) hypertext

The details of the mapping from the concrete hypertext to the visible presentation of the hypertext for the reader are specified here.[4] However, user interface details, such as the positioning and sizing of windows, are orthogonal to the r-model, as discussed later in this report.

---

[3] See also the comparison with anchors that follows in section 3.1.2.

[4] "Visible presentation" is a simplification, since the presentation is not limited to being visible. For example, it might be audible, etc.

2.3.1 Visible hypertext level

An assumption in the r-model is that the underlying hypertext is to be permitted to be used in a distributed environment. The visible hypertext level reflects this assumption. Each **visible HT segment** is associated with a separate user and display. Each segment presents one or more of the active concrete windows to its viewer. The model does not prevent the display of a particular concrete window in more than one segment. Whether (and how) the effects of user interactions to one display may affect what is shown on other user displays is a property of the hypertext model, and not of the r-model.

3 Issues in application of the r-model

We now turn our attention to three aspects of the r-model, which we shall consider in detail. In Section 3.1, we discuss some important components of hypertext systems and how they fit into the r-model. In Section 3.2, we turn our attention to central issues in implementation of a hypertext system that are orthogonal to our model-centered r-model. Finally, in Section 3.3, we discuss the intersection of our r-model with already-existing defined and defacto standards.

3.1 Further discussion of elements of the r-model

A number of structures and components have been identified for hypertexts.[5] Here, we present some of these hypertext elements and describe their categorization within our reference model.

3.1.1 Hypertext model structures

We emphasize that the hypertext's abstract structure is arbitrary in form within the reference model. It may be graph-based, describing only object interrelationships, or it may also have automaton semantics. It need not be homogeneous in form; heterogeneous structures may be appropriate for some applications. It need not be static in form but may be dynamic. Indeed, it need not be explicitly computed or represented. What is required, however, is that it be possible to intuit where it is possible to include content in the hypertext and also the relationships between elements of the content.

3.1.2 Anchors

In some other models of hypertext, anchors have been identified as separatable component of a hypertext.[6] The anchor represents the terminating point or points of a link. In one general form, anchors may be associated with both the source and the target of a one-directional link in a hypertext. They present the relationship between the identified portion of the source and the identified portion of the target. In other implementations, anchors are only associated with source, with the target being the node as a whole. In our Trellis implementations, anchors may or may not be associated with the source—when no anchor is associated with a source then the link is represented by a (graphical) button in a separately displayed palette.

---

[5] See [LSK88], for example, for definitions of related terminology.

[6] See, for example, the Dexter reference model [HS90].

Within the r-model, the display of anchors in source and target is specified in the mapping that defines the concrete content (concrete context level). Both the form of the display and also its position are described here. Issues involving positioning of the target content's display when a link is followed are addressed in the definition of the concrete windows (concrete hypertext level).

3.1.3 Different flavors of links

A hypertext implementation may contain several different kinds of links, each with a different implemented action on selection. The distinction between the different types of link is reflected in the r-model by a difference between the types of their corresponding abstract buttons.

The display of the source or target of a link may be static or may be computed. Such displays are described within the mapping that produces the concrete content representation.

In some circumstances selection of a link may cause an apparent change to the displayed content, for example, insertion of the target's content into place in the source. When the content actually changes in form, this is a matter of interest in the concrete content. However, when the content is actually unchanged in form, as is the case when the target material is inserted, this can be described through the display mapping that produces the concrete windows representation.

3.1.4 Dynamic content

Abstract content may be statically defined or it may be computed. It is useful to distinguish separate categories of computed content from one another. One such categorization distinguishes

- Computed content: executor of an algorithm that produces a subsequently static display
- Dynamic content: Dynamic execution of an algorithm: start on node entry, terminate on node exit
- Filtered computation: Continuously-executing filter

3.2 Orthogonal considerations

The r-model is centered around organizing and categorizing the parts of a model of hypertext. Consequently, there are elements of an implementation, as well as elements of some hypertext models, that are not included in the r-model. These will be presented in this section of the report.

3.2.1 Hypertext browsing semantics

We have previously defined a hypertext system's *browsing semantics* [SF89a] as the dynamic properties of a reader's experience when browsing a document; in other words, as the manner in which the information within the hypertext is to be visited and presented. In most cases, browsing semantics are specified by the code that implements the hypertext system. However, it is also possible to develop a hypertext model with variable browsing semantics; for example our Trellis hypertext model permits specification of the hypertext's browsing semantics [FS89b].[7] Although specifiable browsing semantics are in some hypertext models, they are not in all, and so we have decided not to include them directly in the r-model.

---

[7] The behaviors associated with different link types are reflected by their browsing semantics. Consequently, variable browsing semantics are the implementation mechanism for user-defined link types, as well as other browsing behaviors.

Similarly, we have not included the hypertext's dynamic behavior in the r-model. By dynamic behavior, we mean those cases in which a hypertext system traverses the structure without intervention from the reader [SF89b]. Dynamic behavior is distinct from dynamic content, however. As noted above, dynamic content *is* described within the model.

3.2.2 Characteristics of the content

Some hypertext systems may favor an organization in which each piece of content is treated as a small card-sized unit while others favor organizations in which the content is viewed as a long continuous scroll. Such considerations are outside of the scope of the r-model.

3.2.3 Physical-level descriptions and interchange descriptions

If the structure of the implemented hypertext system closely parallels that of the r-model, it will certainly be necessary to define a storage format for those representations that are specified directly as well as a description of the mappings that produce the others. However, the specific design of such storage formats is outside of the scope of the r-model, as is the equally-important design of formats designed to permit interchange between hypertext systems and installations.

3.2.4 User interfaces

Certainly to the reader of a hypertext, the most visible component of the system is its user interface. However, the user interface is also an element of the system not discussed in the r-model. We note that it is possible to associate many different styles of user interface with the same underlying hypertext model.

3.3 Intersection with existing standards

There are two points of intersection between the r-model and existing standards. The first, in the abstract component level, are the abstractions used to define the abstract content. An appropriate standard to consider for text, for example, would be SGML [ISO86]. Similar utility could be made of standards to define graphical material as well as other content objects. It may be necessary, however, to augment these standard representations with additional information describing the potential interactions defined by the concrete-structure and button-structure associations, and as reflected in the concrete content.

The other point of intersection with proposed standards is in the visible hypertext level. Each visible HT segment and user display may be based around a protocol such as that of the X-windows system [SG86]. Other defacto interface standards such as SunTools, OpenLook, Viewpoint, Motif, and NextStep are also applicable at this point.

4 Discussion and conclusions

We have described a meta-model of hypertext, which we call the r-model, that helps to organize the portions of a hypertext model. It is possible that the hypertext model's design will also correspond to the divisions established in the r-model, but it is equally permissible that the relationships be less-clearly drawn in the hypertext model. Furthermore, the implementation of the hypertext system may also correspond directly to the model or again distinct model concepts may be merged in implementation.

In our own work in developing the Trellis hypertext model and prototype implementations, we have tended to reflect the divisions of the r-model strongly in our hypertext model and also to carry these divisions on into our implementation. In essence, our implementation is based on a collection of abstract data types, where the data types correspond to the representations in the r-model. A natural consequence of this retention of separation has been that it is easy to extend the environment in which the implementation resides—for example to consider designs that permit multiple readers to be active in the hypertext at the same time that a writer is modifying it. Moreover the retention of separation between structure, content, and context permits flexible reuse of the hypertext's structure and of the content of the hypertext.

While we believe that direct application of the r-model has benefits in guiding the implementation of a hypertext system, we also believe that a greater understanding of a hypertext model can be gained by casting it into the form of the r-model. It is this increased understanding that we believe is of primary importance outside of the context of our own development.

Acknowledgments

We would like to thank the Hypertext Standardization Workshop program committee, particularly Judi Moline, for comments that helped us to clarify the points of this report. We also would like to thank the participants in the Workshop's Hypertext Reference Model working group, particularly John Leggett, for discussions that helped identify the similarities and differences between this model and the others that have been proposed.

References

[FS89a] Richard Furuta and P. David Stotts. Separating hypertext content from structure in Trellis. In *Proceedings of Hypertext 2*, June 1989. University of York, June 29th and 30th, 1989.

[FS89b] Richard Furuta and P. David Stotts. Programmable browsing semantics in Trellis. In *Hypertext '89 Proceedings*, pages 27–42. ACM, New York, November 1989.

[HS90] Frank Halasz and Mayer Schwartz. The Dexter hypertext reference model, January 1990. These proceedings.

[ISO86] ISO. *Text and Office Systems—Standard Generalized Markup Language*. October 1986. Document Number: ISO 8879-1986(E).

[LSK88] John Leggett, John L. Schnase, and Charles J. Kacmar. Working definitions of hypertext. Technical Report TAMU 88-020, Department of Computer Science, Texas A&M University, October 1988.

[SF89a] P. David Stotts and Richard Furuta. Petri-net-based hypertext: Document structure with browsing semantics. *ACM Transactions on Information Systems*, 7(1):3–29, January 1989.

[SF89b] P. David Stotts and Richard Furuta. Temporal hyperprogramming. Technical Report CS-TR-2349 and UMIACS-TR-89-113, University of Maryland Department of Computer Science and Institute for Advanced Computer Studies, November 1989.

[SG86] Robert W. Scheifler and Jim Gettys. The X Window system. *ACM Transactions on Graphics*, 5(2):79–109, April 1986.

[Sha80] Alan C. Shaw. A model for document preparation systems. Technical Report 80-04-02, University of Washington, Department of Computer Science, Seattle, WA, April 1980.

The Dexter Hypertext Reference Model[*]

Frank Halasz
Xerox PARC
3333 Coyote Hill Rd.
Palo Alto, CA 94304
halasz@xerox.com

Mayer Schwartz
Tektronix Labs
P.O. Box 500, MS 50-662
Beaverton, OR 97077
mayers@tekchips.labs.tek.com

December 7, 1989

Submitted to the NIST Hypertext Standardization Workshop,
Gaithersburg, MD, January 16-18, 1990

Abstract

This paper presents the Dexter hypertext reference model. The Dexter model is an attempt to capture, both formally and informally, the important abstractions found in a wide range of existing and future hypertext systems. The goal of the model is to provide a principled basis for comparing systems as well as for developing interchange and interoperability standards. The model is divided into three layers. The storage layer describes the network of nodes and links that is the essence of hypertext. The runtime layer describes mechanisms supporting the user's interaction with the hypertext. The within-component layer covers the content and structures within hypertext nodes. The focus of the model is on the storage layer as well as on the mechanisms of anchoring and presentation specification that form the interfaces between the storage layer and the within-component and runtime layers, respectively. The model is formalized using Z [19], a specification language based on set theory. The paper briefly discusses the issues involved in comparing the characteristics of existing systems against the model.

---

[*] Acknowledgement: The model described in this paper grew out a series of workshops on hypertext. The following people attended these workshops and were instrumental in the development of the model: Rob Akscyn, Doug Engelbart, Steve Feiner, Frank Halasz, John Leggett, Don McCracken, Norm Meyrowitz, Tim Oren, Amy Pearl, Catherine Plaisant, Mayer Schwartz, Randy Trigg, Jan Walker, and Bill Wieland. The workshops were organized by Jan Walker and John Leggett.

What do hypertext[1] systems such as NoteCards [10], Neptune [4], KMS [1], Intermedia [23] and Augment [6] have in common? How do they differ? In what way do these systems differ from related classes of systems such as multimedia database systems. At a very abstract level, each of these hypertext systems provides its users with the ability to create, manipulate, and/or examine a network of information-containing nodes interconnected by relational links. Yet these systems differ markedly in the specific data models and sets of functionality that they provide to their users. Augment, Intermedia, NoteCards, and Neptune, for example, all provide their users with a universe of arbitrary-length documents. KMS and Hypercard, in contrast, are built around a model of a fixed-size canvas onto which items such as text and graphics can be placed. Given these two radically different designs, is there anything common between these systems in their notions of hypertext nodes?

In an attempt to provide a principled basis for answering these questions, this paper presents the Dexter hypertext reference model. The model provides a standard hypertext terminology coupled with a formal model of the important abstractions commonly found in a wide range of hypertext systems Thus, the Dexter model serves as a standard against which to compare and contrast the characteristics and functionality of various hypertext (and non-hypertext) systems. The Dexter model also serves as a principled basis on which to develop standards for interoperability and interchange among hypertext systems.

The Dexter reference model described in this paper was initiated as the result of two small workshops on hypertext. The first workshop was held in October, 1988 at the Dexter Inn in New Hampshire. Hence the name of the model. The workshops had representatives from many of the major existing hypertext systems[2]. A large part of the discussion at these workshops was the elicitation of the abstractions common to the major hypertext systems. The Dexter model is an attempt to capture, fill-out, and formalize the results of these discussions.

---

[1] The terms hypertext and hypermedia are often differentiated, with hypertext referring to text-only systems and hypermedia refering to systems that support multiple media. This distinction is not made in the present paper; the term hypertext is used generically to refer to both text-only and multimedia systems.

[2] Participants in the two workshops are listed in the acknowledgements on the first page of this paper.

Among the systems that were discussed at the workshops were: Augment, Concordia/Document Examiner, IGD, FRESS, Intermedia, Hypercard, Hyperties, KMS/ZOG, Neptune/HAM, NoteCards, the Sun Link Service, and Textnet.

Another important focus of the workshops was an attempt to find a common terminology for the hypertext field. This turned out to be an extremely difficult task, especially so in the absence of an understanding of the common (and differing) abstractions among the various systems. The term "node" turned out to be especially difficult given the extreme variation in the use of the term across the various systems. By providing a well-defined set of named abstractions, the Dexter model provides a solution to the hypertext terminology problem. It does so, however, at some cost. In order to avoid confusion, the model does not use contentious terms such as "node", prefering neutral terms such as "component" for the abstraction in the model.

In the present paper, the Dexter model is formulated in Z [19], a formal specification language based on typed set theory. The use of Z provides a rigorous basis for defining the necessary abstractions and for discussing their use and interrelationships. Although an understanding of the Z language is a prerequisite for fully understanding the details of the Dexter model as described in this paper, the paper attempts to provide a complete description of the model in the prose accompanying the formal specification. Readers unfamiliar with Z should be able to gain a full, if not precisely detailed, understanding of the model.

This paper also refers in passing to architectural concepts found in a number of existing hypertext systems including Augment [6], Concordia/Document Examiner [22], Hypercard [8], Hyperties [18], IGD [7], Intermedia [23], KMS [1], Neptune/HAM [4], NoteCards [10], the Sun Link Service [17], and Textnet [20]. The reader is assumed to be familiar with the general characteristics and functionality of these systems. Appropriate background material on these systems can be found in Conklin [3] and in the proceedings of the Hypertext 87 [11] and Hypertext 89 [12] conferences.

This paper is divided in 4 main sections. The first section provides a brief discursive overview of the entire model. The second section describes the storage layer of the model, both formally and informally. The third section describes the runtime layer of the model in a similar manner. The final section discusses issues involved in comparing existing systems against the model.

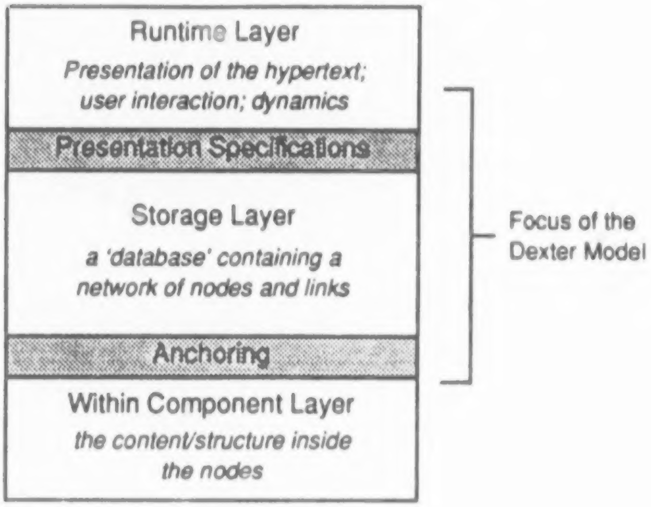

Runtime Layer
Presentation of the hypertext; user interaction; dynamics
Presentation Specifications
Storage Layer
a 'database' containing a network of nodes and links
Anchoring
Within Component Layer
the content/structure inside the nodes
Focus of the Dexter Model

Figure 1: Layers of the Dexter model.

1 An Overview of the Model

The Dexter model divides a hypertext system into three layers, the *runtime* layer, the *storage* layer and the *within-component* layer, as illustrated in Figure 1. The main focus of the model is on the storage layer, which models the basic node/link network structure that is the essence of hypertext. The storage layer describes a 'database' that composed of a hierarchy of data-containing "components" which are interconnected by relational "links". Components correspond to what is typically thought of as nodes in a hypertext network: *cards* in NoteCards and HyperCard, *frames* in KMS, *documents* in Augment and Intermedia, or *articles* in Hyperties. Components contain the chunks of text, graphics, images, animations, etc. that form the basic content in the hypertext network.

The storage layer focuses on the mechanisms by which the components and links are "glued together" to form hypertext networks. The components are treated in this layer as generic containers of data. No attempt is made to model any structure within the container. Thus, the storage layer makes no differentiation between text components and graphics components. Nor does it provide any mechanisms for dealing with the well-defined structure inherent within a structured document (e.g., an ODA document) component.

In contrast, the within-component layer of the model is specifically concerned with the contents and structure *within* the components of the hypertext network. This layer is purposefully not elaborated within the Dexter model. The range of possible content/structure that can be included in a component is open-ended. Text, graphics, animations, simulations, images, and many more types of data have been used as components in existing hypertext systems. It would be folly to attempt a generic model covering all of these data types. Instead, the Dexter model treats within-component structure as being outside of the hypertext model *per se*. It is assumed that other reference models designed specifically to model the structure of particular applications, documents, or data types (ODA, IGES, etc) will be used in conjunction with the Dexter model to capture the entirety of the hypertext, including the with-component content and structure.

An extremely critical piece of the Dexter model, however, is the interface between the hypertext network and the within-component content and structure. The hypertext system requires a mechanism for addressing (refering to) locations or items *within* the content of an individual component. In the Dexter model, this mechanism is know as *anchoring*. The anchoring mechanism is necessary, for example, to support span-to-span links such as are found in Intermedia. In Intermedia, the components are complete structured documents. Links are possible not only between documents, but between spans of characters within one document and spans of characters within another document. Anchors are a mechanism that provides this functionality while maintaining a clean separation between the storage and within-component layers.

The storage and within-component layers treat hypertext as an essentially passive data structure. Hypertext systems, however, go far beyond this in the sense that they provide tools for the user to access, view, and manipulate the network structure. This functionality is captured by the runtime layer of the model. As in the case of within-component structure, the range of possible tools for accessing, viewing, and manipulating a hypertext networks is far too broad and too diverse to allow a simple, generic model. Hence the Dexter model provides only a bare-bones model of the mechanism for presenting a hypertext to the user for viewing and editing. This presentation mechanism captures the essentials of the dynamic, interactional aspects of hypertext systems, but it does not attempt to cover the details of user interaction with the hypertext.

As in the case of anchoring, a critical aspect of the Dexter model is the

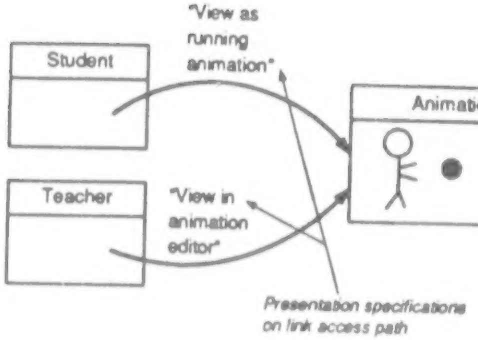

"View as running animation"
Student
Animation
Teacher
"View in animation editor"
Presentation specifications on link access path

Figure 2: Illustration of the need for presentation specifications on the access path (i.e., links) as well as on the components themselves.

interface between the storage layer and the runtime layer. In the Dexter model this is accomplished using the notion of *presentation specifications.* Presentation specifications are a mechanism by which information about how a component/network is to be presented to the user can be encoded into the hypertext network at the storage layer. Thus, the way in which a component is presented to the user can be a function not only of the specific hypertext tool that is doing the presentation (i.e., the specific runtime layer), but can also be a property of the component itself and/or of the access path (link) taken to that component.

Figure 2 illustrates the importance of the presentation specifications mechanism. In this figure, there is an animation component taken from a computer-based training hypertext. This animation component can be accessed from two other components, a "teacher" component and a "student" component. When following the link from the student component, the animation should be brought up as a running animation. In contast, when coming from the teacher component, the animation should be brought up in editing mode ready to be altered. In order to separate these two cases, the runtime layer needs to access presentation information encoded into the links in the network. Presentation specifications are a generic way of doing just this. Like anchoring, it is an interface that allows the storage layer to communicate in generic way with the runtime layer without violating the separation between the two layers.

Figure 3 attempts to give a flavor of the various layers of the Dexter model as they are embedded within an typical hypertext system. The fig-

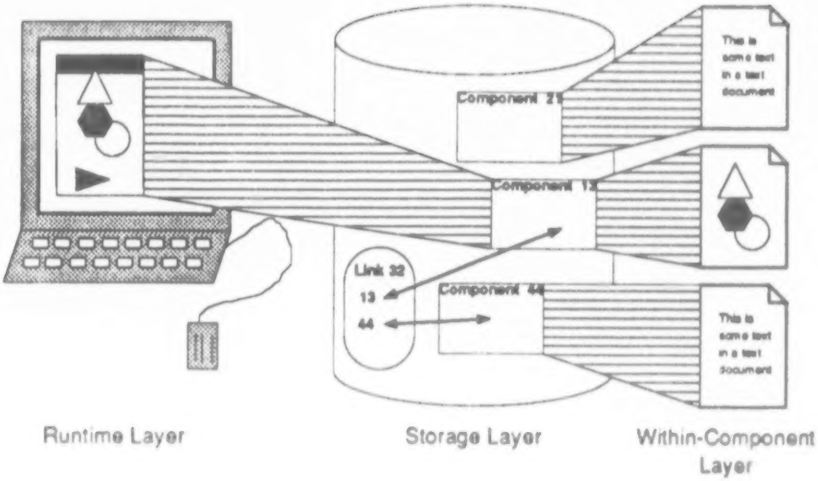


Figure 3: A depiction of the three layers of the Dexter model as embedded in an actual hypertext system.

ure depicts a 3 node/1 link hypertext network. The storage layer contains four entities: the three components (i.e., nodes) and the link. The actual contents (text and graphics) for the components are located to the right of the storage layer in the within-components layer. In the runtime layer, the single graphics component is being presented to the user. The link emanating from this node is marked by an arrowhead located near the bottom of the node's window on the computer screen.

2 Simple Storage Layer Model

2.1 An Overview of the Storage Layer

The storage layer describes the structure of a hypertext as a finite set of components together with two functions, a resolver function and an accessor function. The accessor and resolver functions are jointly responsible for "retrieving" components, i.e., mapping specifications of components into the components themselves.

The fundamental entity and basic unit addressability in the storage layer is the component. A component is either an atom, a link, or a composite entity made up from other components. Atomic components are primitive in the (storage layer of the) model. Their substructure is the concern of the within-components layer. Atomic components are what is typically thought of a "node" in a hypertext system, e.g., a *card* in NoteCards, a *frame* in KMS, a *document* in Intermedia, a *statement* in Augment. Links are entities that represent relations between other components. They are basically a sequence of 2 or more "endpoint specifications" each of which refers to (a part of) a component in the hypertext. The structure of links will be detailed below. Composite components are constructed out of other components. The composite component hierarchy created when one composite component contains another composite is restricted to be a direct-acyclic graph (DAG), i.e., no composite may contain itself either directly or indirectly. Composite components are relative rare in the current generation of hypertext systems. One exception is the Augment system where a document is a tree-structured composition of atomic components called statements.

Every component has a globally unique identity which is captured by its unique identifier (UID). UIDs are primitive in the model, but they are assumed to be uniquely assigned to components across the entire universe of discourse (not just within the context of a single hypertext). The accessor function of the hypertext is responsible for "accessing" a component given its UID, i.e., for mapping a UID into the component "assigned" that UID.

UIDs provide a guaranteed mechanism for addressing any component in a hypertext. But the use of UIDs as a basic addressing mechanism in hypertext may be too restrictive. For example, it is possible in the Augment system to create a link to "the statement containing the word 'pollywog'". The statement specified by this link may not exist or it may change over time as documents are edited. Therefore, the link cannot rely on a specific statement UID to address the target statement. Rather, when the link is followed, the specification must be "resolved" to a UID (if possible), which then can be used to access the correct component.

This kind of indirect addressing is supported in the storage layer using *component specifications* together with the resolver function. The resolver function is responsible for "resolving" a component specification into a UID, which can then be fed to the accessor function to retrieve the specified component. Note, however, that the resolver function is only a partial function. A given specification may not be resolvable into a UID, i.e., the component being specified may not exist. However, it is the case that for every component there is at least one specification that will resolve to the UID for that component. In particular, the UID itself may be used as a specifier, in which case the resolver function is the identity function.

Implementing span-to-span links (e.g., in Intermedia) requires more than simply specifying entire components. Span-to-span linking depends on a mechanism for specifying substructure within components. But in order to preserve the boundary between the hypertext network *per se* and the content/structure within the components, this mechanism cannot depend in any way on knowledge about the internal structure of (atomic) components. In the Dexter model. this is accomplished by an indirect addressing entity called an *anchor*. An anchor has two parts: an *anchor id* and an *anchor value*. The anchor value is an arbitrary value that specifies some location, region, item, or substructure within a component. This anchor value is interpretable only by the applications responsible for handling the content/structure of the component. It is primitive and unrestricted from the viewpoint of the storage layer. The anchor id is an identifier which uniquely identifies its anchor within the scope of its component. Anchors can therefore be uniquely identified across the whole universe by a component UID, anchor id pair.

The two part composition of anchor is designed to provide a fixed point of reference for use by the storage layer, the anchor id, combined with a variable field for use by the within-component layer, the anchor value. As a component changes over time (e.g., when it is edited within the runtime layer), the within-component application will change the anchor value to reflect changes to the internal structure of the component or to reflect within component movement of the point, region, or items to which the anchor is conceptually attached. The anchor is, however, will remain constant, providing a fixed referent that can be used to specify a given structure within a component.

The mechanism of the anchor id can be combined with the component specification mechanism to provide a way of specifying the endpoints of a link. In the model, this is captured by an entity called a *specifier* which consists of a component specification, an anchor id, and two additional fields: a *direction* and a *presentation specification*. A specifier specifies a component and an anchor 'point' within a component that can serve as the endpoint of a link. The direction encodes whether the specified endpoint is to be considered a source of a link, a destination of a link, both a source and a destination, or neither a source nor a destination. (These are encoded by direction values of FROM, TO, BIDIRECT, and NONE, respectively.) The present specification is a primitive value that forms part of the interface between the storage layer and the runtime layer. The nature and use of

BEST COPY AVAILABLE

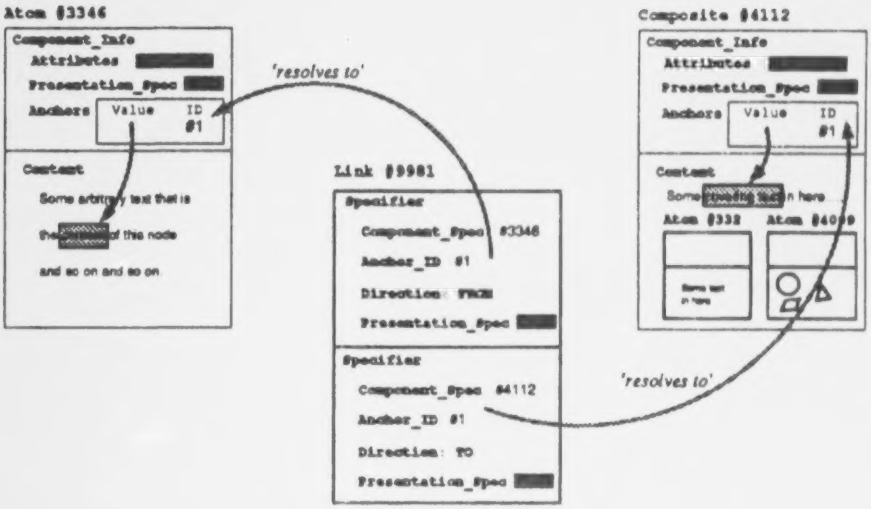


Figure 4: A depiction of overall organization of the storage layer including specifiers, links, and anchors.

present specifications will be discussed in conjunction with the runtime layer below.

Returning to the issue of link components, it is now possible to describe their structure a bit more precisely. In particular, a link is simply a sequence of 2 or more specifiers. Note that this provides for links of arbitrary arity, despite the fact that binary links are standard in existing hypertext systems. Directional links, also standard in existing systems, are handled using the direction field in the specifier.

Figure 4 depicts the overall organization of the storage layer including specifiers, links, and anchors. The figure depicts 5 components including 3 atomic components, 1 composite component (that constructed from two of the atomic components plus some text), and 1 link component that represents a connection from the anchor (i.e., span) within an atomic component (#3346) to the anchor (span) in the composite component (#4112).

In the foregoing discussion, components were described as being either a atom, a link, or a composition of other components. In actuality, this describes what the model calls a *base component*. In contrast, *components* in the model are complex entities that contain a base component together with some associated *component information*. The component information describes the properties of the component other than its 'content'. Specifically, the component information contains a sequence of anchors that index into the component, a present specification that contains information for the runtime layer about how the component should be presented to the user, and a set of arbitrary attribute/value pairs. The attribute/value pairs can be used to attach any arbitrary property (and its value) to a component. For example, keywords can be attached to a component using mutiple 'keyword' attributes. Similarly, a component type system can be implemented in the model by adding to each component a 'type' attribute with an appropriate type specification as its value.

In addition to a data model, the storage layer defines a small set of operations that can be used to access and/or modify a hypertext. All of these operations are defined in such a way as to maintain the invariants of the hypertext, e.g., the fact that the composition hierarchy of components/subcomponents is acyclic. The operations defined in the model include adding a component (atomic, link or composite) to a hypertext, deleting a component from the hypertext, and modifying the contents or ancilliary information (e.g., anchors or attributes) of a component. There are also operatons for retrieving a component given its UID or any specifier that can be resolved to its UID. Finally, there is one operation needed for determining the interconnectivity of the network structure. This operation, *linksToAnchor*, returns the set of links that refer to an anchor when given the anchor and its containing component.

2.2 Formalization of the Storage Layer

As described above, we envision a hypertext system consisting of a set of components, each of which has a UID from the given set *UID*.

[*UID*]

Retrieving a component involves finding its UID and then using that UID to get hold of the actual component; this is accomplished by means of an *accessor* function which returns a component given its UID. UIDs are normally not meant to be visible to clients of a hypertext system. Given a component specification, it may be possible to find the UID to which the component specification refers, by means of a *resolver* function. Component specifications arise from the given set *COMPONENT_SPEC*. We also have a description for the visual presentation (present spec) of a component, which as part of a component is used in the run-time layer but

BEST COPY AVAILABLE

not in the storage layer; these visual descriptions come from the given set $PRESENT\_SPEC$.

$$[COMPONENT\_SPEC, PRESENT\_SPEC]$$

Links are an important kind of component and are supported in every hypertext system. Directionality is sometimes important for links, while at other times it immaterial. We introduce $DIRECTION$ as a free type to model respectively the end of a link as a source, as a destination, as both a source and destination, or as neither.

$$DIRECTION ::= FROM \mid TO \mid BYDIRECT \mid NONE$$

The schema type $SPECIFIER$ essentially takes the form of the description of one end of a "link." This description is sometimes sufficient to determine the UID of the component at one end of a link. As described in the overview, anchoring plays an important part in the model. Anchors are identified by means of a unique (to a component) anchor id from the given set $ANCHOR\_ID$. Anchor values come from the given set $ANCHOR\_VALUE$. Anchors are then just pairs of anchor id and associated anchor value.

$$[ANCHOR\_ID, ANCHOR\_VALUE]$$

$$ANCHOR == ANCHOR\_ID \times ANCHOR\_VALUE$$

A value of type $SPECIFIER$ describes a single end of a link. We include the variable *presentSpec* in the $SPECIFIER$ schema so we can model different ways of visually showing links as we follow them (based on the specifier used), as illustrated in the example shown in Figure 2.

$$
\begin{array}{|l}
\hline
SPECIFIER \\
\textit{componentSpec} : COMPONENT\_SPEC \\
\textit{anchorSpec} : ANCHOR\_ID \\
\textit{presentSpec} : PRESENT\_SPEC \\
\textit{direction} : DIRECTION \\
\hline
\end{array}
$$

Links must include at least two specifiers. What appear to be one-way links, such as Hypercard buttons, can be modeled as two-way links with the button end having a $DIRECTION$ with value $NONE$ and the other end having a $DIRECTION$ with value TO. The two *specifiers* link constraint simplifies the hypertext model. On the other hand there is no reason not to have multi-way links, and so the model accomodates them. In the most general model, duplicate specifiers are allowed. The only constraint is that at least one specifier have a direction of *TO*.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{LINK}} \\
\hline
\mathit{specifiers} : \mathrm{seq}\, \mathit{SPECIFIER} \\
\hline
\#\mathit{specifiers} \geq 2 \\
\exists\, s : \mathrm{ran}\, \mathit{specifiers} \bullet s.\mathit{direction} = \mathit{TO} \\
\hline
\end{array}$$

A base component (a generalization of the traditional "node" or "link") of a hypertext can either be

- an atomic element which is modeled by the given type *ATOM*,

  $[\mathit{ATOM}]$

  models a "node" of a typical hypertext system but with the internal detail omitted.

- a *link* which is modeled by the *LINK* schema given above, or
- a *composite* which can be described recursively as a sequence of base components.

Components can have ancillary information associated with them, such as attribute/value pairs, anchors, or presentation information. Most hypertext systems allow for attributes of components. These attributes can be thought of as attribute/value pairs which can be modeled as a partial function mapping attributes to values. We thus introduce two additional given sets, one for the set of attribute names and the other for the set of possible values:

$[\mathit{ATTRIBUTE}, \mathit{VALUE}]$

The additional information associated with a base component, which was mentioned above, can be captured in the following schema. We include the invariant that anchor ids are unique within a given component, i.e., the number of anchors within a component is equal to the size of the set of (different) anchors within the component.

$$\begin{array}{|l}
\hline
\mathit{COMP\_INFO} \\
\mathit{attributes} : \mathit{ATTRIBUTE} \mathrel{⇻} \mathit{VALUE} \\
\mathit{anchors} : \mathrm{seq}\, \mathit{ANCHOR} \\
\mathit{presentSpec} : \mathit{PRESENT\_SPEC} \\
\hline
\#\mathit{anchors} = \#(\mathit{first}(\!|\mathrm{ran}\ \mathit{anchors}|\!)) \\
\hline
\end{array}$$

Note that a *presentSpec* always has some value. We introduce the function *minInfo* which returns an instance of this schema with "minimal information," that is, no attributes, no anchors and a *presentSpec* which is given as an argument.

$$\begin{array}{|l}
\mathit{minInfo} : \mathit{PRESENT\_SPEC} \rightarrow \mathit{COMP\_INFO} \\
\hline
\forall\, ps : \mathit{PRESENT\_SPEC} \bullet \\
\quad \mathit{minInfo}(ps) = (\mu\, \mathit{info} : \mathit{COMP\_INFO} \mid \\
\qquad \mathit{info}.\mathit{attributes} = \varnothing \wedge \\
\qquad \mathit{info}.\mathit{anchors} = \langle\rangle \wedge \\
\qquad \mathit{info}.\mathit{presentSpec} = ps)
\end{array}$$

We use the recursive type, *BASE_COMPONENT*, to describe the base components of a hypertext system.

$$\begin{array}{lll}
\mathit{BASE\_COMPONENT} & ::= & \mathit{atom}\langle\!\langle \mathit{ATOM} \rangle\!\rangle \\
 & \mid & \mathit{link}\langle\!\langle \mathit{LINK} \rangle\!\rangle \\
 & \mid & \mathit{composite}\langle\!\langle \mathrm{seq}\, \mathit{BASE\_COMPONENT} \rangle\!\rangle
\end{array}$$

Finally, the schema *COMPONENT* represents a base component along with its associated information.

$$\begin{array}{|l}
\hline
\mathit{COMPONENT} \\
\mathit{compBase} : \mathit{BASE\_COMPONENT} \\
\mathit{compInfo} : \mathit{COMP\_INFO} \\
\hline
\end{array}$$

The functions defined in the remainder of this section are there just to make the specification of the model easier to read and understand — they are not meant to have any particular significance in their own right. The following function builds a component given its base component and associated information.

$$
\begin{array}{|l}
component : BASE\_COMPONENT \times COMP\_INFO \\
\qquad\qquad \rightarrow COMPONENT \\
\hline
component = (\lambda\, b : BASE\_COMPONENT;\ i : COMP\_INFO \bullet \\
\quad (\mu\, c : COMPONENT \mid \\
\qquad c.compBase = b \land \\
\qquad c.compInfo = i))
\end{array}
$$

The following two functions extract respectively the base component and associated information of a component.

$$
\begin{array}{|l}
base : COMPONENT \rightarrow BASE\_COMPONENT \\
info : COMPONENT \rightarrow COMP\_INFO \\
\hline
\forall\, c : COMPONENT \bullet \\
\quad base(c) = c.compBase \land \\
\quad info(c) = c.compInfo
\end{array}
$$

We introduce three predicates (prefix relations) which are respectively true iff a component is an atom, a link, or a composite.

$$
\begin{array}{|l}
\mathrm{isAtom}\_ : \mathbb{P}\ COMPONENT \\
\mathrm{isLink}\_ : \mathbb{P}\ COMPONENT \\
\mathrm{isComposite}\_ : \mathbb{P}\ COMPONENT \\
\hline
\forall\, c : COMPONENT \bullet \\
\quad \mathrm{isAtom}\ c \Leftrightarrow base(c) \in \mathrm{ran}\ atom \land \\
\quad \mathrm{isLink}\ c \Leftrightarrow base(c) \in \mathrm{ran}\ link \land \\
\quad \mathrm{isComposite}\ c \Leftrightarrow base(c) \in \mathrm{ran}\ composite
\end{array}
$$

We also define a "type" consistency relationship between components — that is, two components are "type consistent" is they are both atoms, both links, or both composites.

$$
\begin{array}{|l}
\_\mathrm{typeConsistent}\_ : COMPONENT \leftrightarrow COMPONENT \\
\hline
\forall\, c_1, c_2 : COMPONENT \bullet \\
\quad c_1\ \mathrm{typeConsistent}\ c_2 \Leftrightarrow \\
\qquad (\mathrm{isAtom}\ c_1 \land \mathrm{isAtom}\ c_2) \lor \\
\qquad (\mathrm{isLink}\ c_1 \land \mathrm{isLink}\ c_2) \lor \\
\qquad (\mathrm{isComposite}\ c_1 \land \mathrm{isComposite}\ c_2)
\end{array}
$$

Because link components are referred to quite frequently in what follows, we introduce the schema *LinkComp* so we can define variables of that type.

$$
\begin{array}{|l}
\hline
\textit{LinkComp} \\
\mathit{COMPONENT} \\
\hline
\mathit{compBase} \in \operatorname{ran} \mathit{link} \\
\hline
\end{array}
$$

We also introduce some helpful functions to extract the various parts that make up a base component type. The first two functions are only defined for link components and return respectively the set of component specs for the link and the set of anchor ids for the link.

$$
\begin{array}{|l}
\mathit{componentSpecs} : \mathit{LinkComp} \mathrel{+\!\!\!\!\to} \mathbb{F}\ \mathit{COMPONENT\_SPEC} \\
\mathit{anchorSpecs} : \mathit{LinkComp} \mathrel{+\!\!\!\!\to} \mathbb{F}\ \mathit{ANCHOR\_ID} \\
\hline
\forall c : \mathit{LinkComp} \bullet \\
\qquad \mathit{componentSpecs}(c) = \{ \mathit{cs} : \mathit{COMPONENT\_SPEC} \mid \\
\qquad\qquad \exists s : \operatorname{ran}(\mathit{link}^{\sim}(\mathit{base}(c))).\mathit{specifiers} \bullet \\
\qquad\qquad\qquad \mathit{cs} = s.\mathit{componentSpec} \} \land \\
\qquad \mathit{anchorSpecs}(c) = \{ \mathit{as} : \mathit{ANCHOR\_ID} \mid \\
\qquad\qquad \exists s : \operatorname{ran}(\mathit{link}^{\sim}(\mathit{base}(c))).\mathit{specifiers} \bullet \\
\qquad\qquad\qquad \mathit{as} = s.\mathit{anchorSpec} \}
\end{array}
$$

The next two functions are defined for any component and return respectively its attributes and its anchors.

$$
\begin{array}{|l}
\mathit{attributes} : \mathit{COMPONENT} \to (\mathit{ATTRIBUTE} \mathrel{+\!\!\!\!\to} \mathit{VALUE}) \\
\mathit{anchors} : \mathit{COMPONENT} \to \mathbb{F}\ \mathit{ANCHOR} \\
\hline
\forall c : \mathit{COMPONENT} \bullet \\
\qquad \mathit{attributes}(c) = (\mathit{info}(c)).\mathit{attributes} \land \\
\qquad \mathit{anchors}(c) = \operatorname{ran}(\mathit{info}(c)).\mathit{anchors}
\end{array}
$$

Finally, we introduce a function which given a component returns a component just like the given one except that the attributes function is (possibly) overwritten with a new value for a given attribute.

$$
\begin{array}{|l}
\mathit{modifyAttribute} : \mathit{COMPONENT} \times \mathit{ATTRIBUTE} \times \mathit{VALUE} \\
\qquad\qquad\qquad\qquad \to \mathit{COMPONENT} \\
\hline
\mathit{modifyAttribute} = (\lambda\, c : \mathit{COMPONENT};\ a : \mathit{ATTRIBUTE}; \\
\qquad\qquad\qquad\qquad v : \mathit{VALUE} \bullet \\
\quad (\mu\, c' : \mathit{COMPONENT} \mid \exists\, i, i' : \mathit{COMP\_INFO} \mid \\
\qquad\quad i = \mathit{info}(c) \bullet \\
\qquad\quad i'.\mathit{attributes} = i.\mathit{attributes} \oplus \{a \mapsto v\} \land \\
\qquad\quad i'.\mathit{anchors} = i.\mathit{anchors} \land \\
\qquad\quad i'.\mathit{presentSpec} = i.\mathit{presentSpec} \land \\
\qquad\quad c' = \mathit{component}(\mathit{base}(c), i')))
\end{array}
$$

Components can have sub-components and the same component may be a sub-component to more than one component. This relationship will be denoted by _subcomp_ and is defined below.

$$
\begin{array}{|l}
\_\mathrm{subcomp}\_ : \mathit{COMPONENT} \leftrightarrow \mathit{COMPONENT} \\
\hline
\forall\, c_1, c_2 : \mathit{COMPONENT} \bullet \\
\quad c_1 \mathrel{\mathrm{subcomp}} c_2 \Leftrightarrow \\
\qquad \mathit{base}(c_1) \in \mathrm{ran}(\mathit{composite}^{\sim}(\mathit{base}(c_2)))
\end{array}
$$

A hypertext system, modeled by the schema *PROTO_HYPERTEXT*, has three parts. (1) The set of *components* represents the traditional "nodes" and "links" of a hypertext system. (2) A partial function termed the *resolver* returns the UID for a given component specifier. Note that more than one specifier may return the same UID. (3) To actually get hold of a component, we introduce an *accessor* function which given a UID returns a component. Note that this function while partial, is invertible.

$$
\begin{array}{|l}
\hline
\mathit{PROTO\_HYPERTEXT} \\
\hline
\mathit{components} : \mathbb{F}\ \mathit{COMPONENT} \\
\mathit{resolver} : \mathit{COMPONENT\_SPEC} \rightharpoonup \mathit{UID} \\
\mathit{accessor} : \mathit{UID} \rightarrowtail\!\!\!\!\!\!\!\!\;\;+\;\; \mathit{COMPONENT} \\
\hline
\end{array}
$$

To identify those links resolving to a given component, we introduce the function *linksTo* which, given a hypertext system and the UID of a component in the system, returns the UIDs of links resolving to that component.

$$
\begin{array}{|l}
linksTo : PROTO\_HYPERTEXT \times UID \rightarrow \mathbb{F}\ UID \\
\hline
linksTo = (\lambda\, H : PROTO\_HYPERTEXT;\ u : UID \bullet \{uid : UID \mid \\
\quad (\exists\, comp : LinkComp \mid comp \in H.components \bullet \\
\qquad uid = H.accessor^{\sim}(comp) \wedge \\
\qquad (\exists\, s : COMPONENT\_SPEC \mid \\
\qquad\quad s \in componentSpecs(comp) \bullet \\
\qquad\quad u = H.resolver(s)))\})
\end{array}
$$

There are four constraints which must be satisfied by an instance of the schema *PROTO_HYPERTEXT* before we can call it a *HYPERTEXT*.

- The *accessor* function must yield a value for every component. Because this function is invertible, every component must then have a UID.
- The *resolver* function must be able produce all possible valid UIDs.
- There are no cycles in the component-subcomponent relationship, that is no component may be a subcomponent (directly or transitively) of itself.
- The anchor ids of a component must be the same as the anchor ids of the component specifiers of the links resolving to the component.

$$
\begin{array}{|l}
\multicolumn{1}{l}{HYPERTEXT} \\
\hline
PROTO\_HYPERTEXT \\
\hline
\forall\, c : components \bullet c \in \mathrm{ran}\ accessor \\
\mathrm{ran}\ resolver = \mathrm{dom}\ accessor \\
\forall\, c : components \bullet (c, c) \notin (\_\mathrm{subcomp}\_)^{*} \\
\forall\, c : components \bullet \exists\, lids : \mathbb{F}\ UID \mid \\
\quad lids = linksTo(\theta PROTO\_HYPERTEXT, accessor^{\sim}(c)) \bullet \\
\quad first (\!| anchors(c) |\!) = \\
\qquad \bigcup((anchorSpecs \circ accessor)(\!| lids |\!)) \\
\hline
\end{array}
$$

2.3 Adding New Components

In this section the model adding a new component to a hypertext. The last function defined in this section, *CreateNewComponent*, is the function actually called from the run-time layer and is also part of the external view of the model. (See the section on conformance with the reference model for more about this external view.)

Adding a new component to the hypertext is given by the following function. It ensures that the range of the accessor function is extended to include the new component. The resolver function is also extended so that there is at least one specifier for the new component's corresponding UID.

$$
\begin{array}{|l}
\mathit{createComponent} : \mathit{HYPERTEXT} \times \mathit{COMPONENT} \\
\qquad\qquad\qquad\qquad\qquad \rightarrow \mathit{HYPERTEXT} \\
\hline
\forall H : \mathit{HYPERTEXT};\ c : \mathit{COMPONENT} \bullet \\
\quad \exists H' : \mathit{HYPERTEXT} \mid \\
\qquad H'.\mathit{components} = H.\mathit{components} \cup \{c\} \land \\
\qquad (\exists_1\ \mathit{uid} : \mathit{UID} \bullet \\
\qquad\quad (\exists\ \mathit{componentSpec} : \mathit{COMPONENT\_SPEC} \bullet \\
\qquad\qquad H'.\mathit{accessor} = H.\mathit{accessor} \cup \{\mathit{uid} \mapsto c\} \land \\
\qquad\qquad H'.\mathit{resolver} = H.\mathit{resolver} \cup \\
\qquad\qquad\quad \{\mathit{componentSpec} \mapsto \mathit{uid}\})) \bullet \\
\qquad \mathit{createComponent}(H, c) = H'
\end{array}
$$

The functions for creating a new node, link, and composite respectively are given below. They use the function *createComponent* described above.

$$
\begin{array}{|l}
\mathit{createAtomicComponent} : \mathit{HYPERTEXT} \times \mathit{ATOM} \\
\quad \times \mathit{PRESENT\_SPEC} \rightarrow \mathit{HYPERTEXT} \times \mathit{COMPONENT} \\
\hline
\forall H : \mathit{HYPERTEXT};\ a : \mathit{ATOM};\ ps : \mathit{PRESENT\_SPEC} \bullet \\
\quad \exists c : \mathit{COMPONENT} \mid c = \mathit{component}(\mathit{atom}(a), \mathit{minInfo}(ps)) \bullet \\
\qquad \mathit{createAtomicComponent}(H, a, ps) = \\
\qquad\qquad (\mathit{createComponent}(H, c), c)
\end{array}
$$

In creating a link, we must ensure that all of its component specifiers resolve to existing components. To test for such consistency among links we introduce the following link consistency predicate as a prefix relation.

$$\begin{array}{|l}
\text{linkConsistent}\_ : \mathbb{P}\ HYPERTEXT \\
\hline
\forall H : HYPERTEXT \bullet \\
\quad \text{linkConsistent}\ H \Leftrightarrow \\
\qquad (\forall l : LINK;\ s : SPECIFIER \mid \\
\qquad\quad (\exists cl : LinkComp \mid cl \in H.components \bullet \\
\qquad\qquad l = link^{\sim}(base(cl))) \wedge \\
\qquad\quad s \in \operatorname{ran} l.specifiers \bullet \\
\qquad\quad (\exists c : COMPONENT \mid c \in H.components \bullet \\
\qquad\qquad (H.accessor \circ H.resolver)(s.componentSpec) = c))
\end{array}$$

Creating a new link component is then given by the following function.

$$\begin{array}{|l}
createLinkComponent : HYPERTEXT \times LINK \times PRESENT\_SPEC \\
\qquad\qquad \rightarrow HYPERTEXT \times COMPONENT \\
\hline
\forall H : HYPERTEXT;\ l : LINK;\ ps : PRESENT\_SPEC \bullet \\
\quad \exists H' : HYPERTEXT;\ c : COMPONENT \mid \\
\qquad c = component(link(l), minInfo(ps)) \wedge \\
\qquad H' = createComponent(H, c) \wedge \\
\qquad createLinkComponent(H, l, ps) = (H', c) \bullet \\
\qquad \text{linkConsistent}\ H'
\end{array}$$

In creating a composite we must ensure that any subcomponents of the new composite are already in the hypertext.

$$\begin{array}{|l}
createCompositeComponent : \\
\quad HYPERTEXT \times \operatorname{seq} BASE\_COMPONENT \\
\qquad \times PRESENT\_SPEC \rightarrow HYPERTEXT \times COMPONENT \\
\hline
\forall H : HYPERTEXT;\ s : \operatorname{seq} BASE\_COMPONENT; \\
\qquad\qquad ps : PRESENT\_SPEC \bullet \\
\quad \exists newComp : COMPONENT \mid \\
\qquad newComp = component(composite(s), minInfo(ps)) \bullet \\
\qquad createCompositeComponent(H, s, ps) = \\
\qquad\quad (createComponent(H, newComp), newComp) \wedge \\
\qquad (\forall c : COMPONENT \mid base(c) \in \operatorname{ran} s \bullet \\
\qquad\quad c \in H.components)
\end{array}$$

We package creating a new component with the following function. This is the function which will ultimately be invoked from the run-time layer.

$$
\begin{array}{|l}
CreateNewComponent : HYPERTEXT \times BASE\_COMPONENT \\
\qquad\quad \times PRESENT\_SPEC \rightarrow HYPERTEXT \times COMPONENT \\
\hline
\forall H : HYPERTEXT;\ bc : BASE\_COMPONENT; \\
\qquad\qquad ps : PRESENT\_SPEC \bullet \\
\quad ((\exists a : ATOM \bullet bc = atom(a)) \Rightarrow \\
\qquad CreateNewComponent(H, bc, ps) = \\
\qquad\quad createAtomicComponent(H, atom^{\sim}(bc), ps)) \land \\
\quad ((\exists l : LINK \bullet bc = link(l)) \Rightarrow \\
\qquad CreateNewComponent(H, bc, ps) = \\
\qquad\quad createLinkComponent(H, link^{\sim}(bc), ps)) \land \\
\quad ((\exists s : \mathrm{seq}\ BASE\_COMPONENT \bullet bc = composite(s)) \Rightarrow \\
\qquad CreateNewComponent(H, bc, ps) = \\
\qquad\quad createCompositeComponent(H, composite^{\sim}(bc), ps))
\end{array}
$$

2.4 Deleting A Component

In deleting a component we must ensure that we remove any links whose specifiers resolves to that component.

$$
\begin{array}{|l}
DeleteComponent : HYPERTEXT \times UID \rightarrow HYPERTEXT \\
\hline
DeleteComponent = (\lambda\, H : HYPERTEXT;\ uid : UID \bullet \\
\quad (\mu\, H' : HYPERTEXT \mid \exists\, uids : \mathbb{F}\ UID \mid \\
\qquad uids = \{uid\} \cup linksTo(H, uid) \bullet \\
\qquad\quad H'.components = H.components \setminus H.accessor(\!|uids|\!) \land \\
\qquad\quad H'.accessor = uids \lhd\!\!\!- H.accessor \land \\
\qquad\quad H'.resolver = H.resolver \rhd\!\!\!- uids))
\end{array}
$$

2.5 Modifying Components

In modifying a component we require that its associated information remain unchanged, that its type (atom, link, or composite) remain unchanged, and that the resulting hypertext remains link consistent.

$$
\begin{array}{|l}
\mathit{ModifyComponent} : \mathit{HYPERTEXT} \times \mathit{UID} \times \mathit{COMPONENT} \\
\qquad\qquad\qquad\qquad \rightarrow \mathit{HYPERTEXT} \\
\hline
\forall H : \mathit{HYPERTEXT};\ \mathit{uid} : \mathit{UID};\ c' : \mathit{COMPONENT} \bullet \\
\quad \exists c : \mathit{COMPONENT};\ H' : \mathit{HYPERTEXT} \mid \\
\qquad c = H.\mathit{accessor}(\mathit{uid}) \land \\
\qquad H'.\mathit{components} = H.\mathit{components} \setminus \{c\} \cup \{c'\} \land \\
\qquad H'.\mathit{accessor} = H.\mathit{accessor} \oplus \{\mathit{uid} \mapsto c'\} \land \\
\qquad H'.\mathit{resolver} = H.\mathit{resolver} \land \\
\qquad \mathit{info}(c') = \mathit{info}(c) \land \\
\qquad c\ \mathrm{typeConsistent}\ c' \land \\
\qquad \mathrm{linkConsistent}\ H' \bullet \\
\quad \mathit{ModifyComponent}(H, \mathit{uid}, c) = H'
\end{array}
$$

2.6 Retrieving A Component

To retrieve a component, given its UID, means just to have the returned value of the *accessor* function.

$$
\begin{array}{|l}
\mathit{getComponent} : \mathit{HYPERTEXT} \times \mathit{UID} \rightarrow \mathit{COMPONENT} \\
\hline
\forall H : \mathit{HYPERTEXT};\ \mathit{uid} : \mathit{UID} \bullet \\
\quad \mathit{getComponent}(H, \mathit{uid}) = H.\mathit{accessor}(\mathit{uid})
\end{array}
$$

Given a UID which happens to represent a link, there exist operations which return either a source or destination specifier for that component.

2.7 Attributes

We introduce functions to both get and set the value of a given attribute (if it exists) for a given component.

$$
\begin{array}{|l}
\mathit{AttributeValue} : \mathit{HYPERTEXT} \times \mathit{UID} \times \mathit{ATTRIBUTE} \rightarrow \mathit{VALUE} \\
\hline
\forall H : \mathit{HYPERTEXT};\ \mathit{uid} : \mathit{UID};\ a : \mathit{ATTRIBUTE} \bullet \\
\quad (\exists c : \mathit{COMPONENT} \mid c = H.\mathit{accessor}(\mathit{uid}) \bullet \\
\qquad \mathit{AttributeValue}(H, \mathit{uid}, a) = \mathit{attributes}(c)(a))
\end{array}
$$

$$\begin{array}{|l}
\mathit{SetAttributeValue} : \mathit{HYPERTEXT} \times \mathit{UID} \times \mathit{ATTRIBUTE} \times \mathit{VALUE} \\
\qquad\qquad\qquad\qquad \rightarrow \mathit{HYPERTEXT} \\
\hline
\mathit{SetAttributeValue} = \\
\quad (\lambda\, H : \mathit{HYPERTEXT};\ \mathit{uid} : \mathit{UID};\ a : \mathit{ATTRIBUTE}; \\
\qquad\quad v : \mathit{VALUE} \bullet \\
\quad (\mu\, H' : \mathit{HYPERTEXT} \mid \exists\, c, c' : \mathit{COMPONENT} \bullet \\
\qquad c = H.\mathit{accessor}(\mathit{uid}) \land \\
\qquad c' = \mathit{modifyAttribute}(c, a, v) \land \\
\qquad H'.\mathit{components} = H.\mathit{components} \setminus \{c\} \cup \{c'\} \land \\
\qquad H'.\mathit{accessor} = H.\mathit{accessor} \oplus \{\mathit{uid} \mapsto c'\} \land \\
\qquad H'.\mathit{resolver} = H.\mathit{resolver}))
\end{array}$$

There is also a function which returns the set of all component attributes.

$$\begin{array}{|l}
\mathit{AllAttributes} : \mathit{HYPERTEXT} \rightarrow \mathbb{F}\ \mathit{ATTRIBUTE} \\
\hline
\forall\, H : \mathit{HYPERTEXT} \bullet \\
\quad \mathit{AllAttributes}(H) = \{a : \mathit{ATTRIBUTE} \mid \exists\, c : \mathit{COMPONENT} \bullet \\
\qquad a \in \mathrm{dom}(\mathit{attributes}(c))\}
\end{array}$$

2.8 Anchors

It is sometimes useful to know the link components which are associated with a particular anchor. The function *LinksToAnchor* returns the set of link component uids associated with a particular anchor id for a particular component id.

$$\begin{array}{|l}
\mathit{LinksToAnchor} : \mathit{HYPERTEXT} \times \mathit{UID} \times \mathit{ANCHOR\_ID} \rightarrow \mathbb{F}\ \mathit{UID} \\
\hline
\mathit{LinksToAnchor} = \\
\quad (\lambda\, H : \mathit{HYPERTEXT};\ u : \mathit{UID};\ \mathit{aid} : \mathit{ANCHOR\_ID} \bullet \\
\qquad \{\mathit{lid} : \mathit{UID} \mid \exists\, \mathit{lids} : \mathbb{F}\ \mathit{UID} \mid \\
\qquad\quad \mathit{lids} = \mathit{linksTo}(H, u) \land \mathit{lid} \in \mathit{lids} \bullet \\
\qquad\quad \mathit{aid} \in (\mathit{anchorSpecs} \circ H.\mathit{accessor})(\mathit{lid})\})
\end{array}$$

3 Simple Runtime Layer Model

3.1 An Overview of the Runtime Layer

The fundamental concept in the runtime layer is the *instantiation* of a component. An instantiation is a presentation of the component to the user. Operationally, an instantiation should be thought of as a kind of runtime cache for the component. A 'copy' of the component is cached in the instantiation, the user views and/or edits this instantiation, and the altered cache is then 'written' back into the storage layer. Note that there can be more than one simultaneous instantiation for any given component. Each instantiation is assigned a unique (within session, see below) instantiation identifier (IID).

Instantiation of a component also results in instantiation of its anchors. An instantiated anchor is known as a *link marker*. This terminology is congruent with that used in Intermedia, where the term "anchor" refers to an attachment point or region and the term "link marker"refers to the visible manifestation of that anchor in a displayed document. In order to accomodate the link marker notion within the model, an instantiation is actually a complex entity containing a *base instantiation* together with a sequence of link markers and a function mapping link markers to the anchors they instantiate. A base instantiation is a primitive in the model that represents some sort of presentation of the component to the user.

At any given moment, the user of a hypertext can be viewing and/or editing any number of component instantiations. The runtime layer includes an entity called a *session* which serves to keep track of the moment-by-moment mapping between components and their instantiations. Specifically, when a user wants to access a hypertext, he or she opens a session on that hypertext. The user can then create instantiations of components in the hypertext (an action known as "presenting" the component). The user can edit these instantiations, can modify the component based on the accumulated edits to the instantiation (an action known as "realizing" the edits), and finally can destroy the instantiation (an action known as "unpresenting" a component). When the user is finished interacting with the hypertext, the session is closed.

In the model, the session entity contains the hypertext being accessed, a mapping from the IIDs of the session's current instantiations to their corresponding components in the hypertext, a history, a runtime resolver function, an instantiator function, and a realizer function. At any given moment, the history is a sequence of all operations carried since the last open session operation. In the present version of the model, this history is used only in defining the notion of a read-only session. It is intended to be available, however, to any operation that needs to be conditionalized on preceeding operations.

The session's runtime resolver function is the runtime version of the storage layer's resolver function. Like the resolver, it maps specifiers into component UIDs. The runtime resolver, however, can use information about the current session, including its history, in the resolution process. The storage resolver layer has no access to such runtime information. For example, a specifier may refer to "the most recently accessed component named 'xyzzy'". The runtime resolver is responsible for mapping this specifier into the UID matching this specification. The storage layer resolver would not be able handle this specification. The runtime resolver is restricted to be a superset of the storage layer resolver function; any specifier that the storage layer resolver can resolve to a UID must be resolved to the same UID by the runtime resolver.

At the heart of the runtime model is the session's *instantiator* function. Input to the instantiator consists of a component (UID) and a presentation specification. The instantiator returns an instantiation of the component as part of the session. The *presentation specification* is primitive in the model, but is intended to contain information specifying how the component being instantiated is to be "presented" by the system during this instantiation. Note that the component itself has a presentation specification from the storage layer of the model. This presentation specification is meant to contain information about the component's own notion of how it should be presented. It is the responsibility of the instantiator function to adjudicate (by selection or combination or otherwise) among the presentation specification passed to the instantiator and the presentation specification attached to the component being instantiated. The model in its current form does not make this adjudication explicit.

The instantiator function is the core of a the *present component* operation. Present component takes a component specifier (together with a session and a presentation specification) and calls the instantiator using the component UID derived from resolving the specifier. Present component in turn is the core of the *follow link* operation. Follow link takes (the IID of) an instantiation together with a link marker contained within that instantiation. It then presents the component(s) that are at the destination endpoints (i.e., endpoints whose specifier has direction of TO) of all link(s) that have as an endpoint the anchor represented by the given link marker. In the case where all links are binary, this is equivalent to following a link from the link marker for its source. The result of following the link is a presentation of its destination component and anchor.

The instantiator function also has an "inverse" function called the *realizer* function which takes an instantiation and returns a (new) component that "reflects" the current state of the instantiation (i.e., including recent edits to the instantiation). This is the basic mechanism for "writing back the cache" after an instantiation has been edited. The component produced by the realizer is used as an argument to the storage layer modify composite operation to replace the component with the edited component. This operation is wrapped in the function called *realize edits* in the runtime layer.

3.2 Formalization of the Runtime Layer

The runtime model depends on the notion of an *instantation* which is the visual representation of some component. Each instantiation has a unique instantiation id from the given set *IID*.

$$[\mathit{IID}]$$

An instantiation consists of a *base instantiation* which "represents" a component, a sequence of *link markers* which "represents" the anchors of the component, and a function mapping link markers to anchor ids.

$$[\mathit{BASE\_INSTANTIATION}, \mathit{LINK\_MARKER}]$$

$$\begin{array}{|l}
\textit{INSTANTIATION} \\
\hline
\mathit{base} : \mathit{BASE\_INSTANTIATION} \\
\mathit{links} : \mathrm{seq}\ \mathit{LINK\_MARKER} \\
\mathit{linkAnchor} : \mathit{LINK\_MARKER} \rightarrow \mathit{ANCHOR\_ID} \\
\hline
\mathrm{dom}\ \mathit{linkAnchor} = \mathrm{ran}\ \mathit{links} \\
\hline
\end{array}$$

A user manipulates instantiations, so that there must be a way of mapping from instantiations to components. The function variable *instants* in the *SESSION* schema defined below maps an instantiation id to a pair consisting of an instantiation and the UID of its corresponding component. The *accessor* function in the *HYPERTEXT* schema then maps these UIDs to components. More than one instantiation may be associated with the same UID and hence with the same component.

A hypertext is manipulated in a session which is model by the *SESSION* schema. The *OPERATION* free type names the various operations a user can perform during a hypertext session.

$$\begin{aligned} OPERATION ::= {} & OPEN \mid CLOSE \\ \mid {} & PRESENT \mid UNPRESENT \\ \mid {} & CREATE \mid EDIT \mid SAVE \mid DELETE \end{aligned}$$

During a session, a user opens up one or more instantiations of hypertext components through which the hypertext may be modified. We use the term *presents* to denote opening up an instantiation on a component because the component is presented to the user by means of the instantiation. Instantiations are not only a function of the component which they represent, and two presentation specifiers — one implicitly from the component's *compInfo* and the other explicitly, either user given or from a link specifier — but also implicitly of the "current" set of instantiations. The function *instantiator* which is part of the schema *SESSION* captures this relationship. In saving the result of a series of edits, the reverse of the *instantiator* function is needed; we call this function a *realizer* function. It takes an instantiation and returns a component based on the current session.

There are some component specifiers which can only be resolved at run-time. An example of such a specifier is "the last node visited." The storage layer should be independent of such component specifiers. We introduce the notion of a run-time resolver which is just an extension of the regular resolver function. Note that the invariants on anchors given in the schema for *HYPERTEXT* only apply to those component specifiers which are in the domain of *H.resolver*. Also the *LinksToAnchor* function will not give those links with component specifiers resolvable only at run-time (not in the domain of *H.resolver*) — these additional links must be captured in the run-time layer.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{SESSION}}\\
\hline
H : \mathit{HYPERTEXT}\\
\mathit{history} : \mathrm{seq}\ \mathit{OPERATION}\\
\mathit{instants} : \mathit{IID} \mathrel{\rightarrowtail\!\!\!\!\!\!+} (\mathit{INSTANTIATION} \times \mathit{UID})\\
\mathit{instantiator} : \mathit{UID} \times \mathit{PRESENT\_SPEC} \rightarrow \mathit{INSTANTIATION}\\
\mathit{realizer} : \mathit{INSTANTIATION} \rightarrow \mathit{COMPONENT}\\
\mathit{runTimeResolver} : \mathit{COMPONENT\_SPEC} \mathrel{\rightarrow\!\!\!\!\!\!+} \mathit{UID}\\
\hline
\mathit{head}(\mathit{history}) = \mathit{OPEN}\\
\forall\, \mathit{uid} : \mathit{UID};\ \mathit{ps} : \mathit{PRESENT\_SPEC} \mid\\
\quad \mathit{uid} \in \mathrm{dom}\, H.\mathit{accessor} \bullet\\
\qquad \mathit{realizer}(\mathit{instantiator}(\mathit{uid}, \mathit{ps})) = H.\mathit{accessor}(\mathit{uid}) \wedge\\
H.\mathit{resolver} \subseteq \mathit{runTimeResolver}\\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\Delta \textit{SESSION}}\\
\hline
\mathit{SESSION}\\
\mathit{SESSION}'\\
\hline
\#\mathit{history}' = \#\mathit{history} + 1\\
\mathit{instantiator}' = \mathit{instantiator}\\
\mathit{realizer}' = \mathit{realizer}\\
\hline
\end{array}
$$

A session begins with an existing hypertext (storage system) and a clean instantiation slate.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{openSession}}\\
\hline
\mathit{SESSION}\\
\mathit{hypertext?} : \mathit{HYPERTEXT}\\
\hline
H = \mathit{hypertext?}\\
\mathit{history} = \langle \mathit{OPEN} \rangle\\
\mathit{instants} = \varnothing\\
\hline
\end{array}
$$

Because there are several operations which can open up a new instantiation, we introduce the following function which opens up a set of new instantiation on an existing set of component.

$$
\begin{array}{|l}
openComponents : \\
\quad SESSION \times \mathbb{F}(SPECIFIER \times PRESENT\_SPEC) \\
\qquad\qquad \to SESSION \\
\hline
\forall S : SESSION;\ specs : \mathbb{F}(SPECIFIER \times PRESENT\_SPEC) \bullet \\
\quad \exists S' : SESSION;\ iids : \mathbb{F}\ IID; \\
\qquad\quad newInstants : IID \rightarrowtail\!\!\!\!\twoheadrightarrow (INSTANTIATION \times UID) \mid \\
\qquad S'.H = S.H \land \\
\qquad S'.runTimeResolver = S.runTimeResolver \land \\
\qquad S'.history = S.history \frown \langle PRESENT \rangle \land \\
\qquad S'.instants = S.instants \oplus newInstants \land \\
\qquad \# iids = \# specs \land iids \cap \mathrm{dom}\ S.instants = \varnothing \land \\
\qquad \mathrm{dom}\ newInstants = iids \land \\
\qquad (\forall s : specs \bullet \\
\qquad\quad \exists iid : iids;\ uid : UID; \\
\qquad\qquad cs : COMPONENT\_SPEC; \\
\qquad\qquad ps : PRESENT\_SPEC; \\
\qquad\qquad inst : INSTANTIATION \mid \\
\qquad\quad cs = (first(s)).componentSpec \land \\
\qquad\quad ps = second(s) \land \\
\qquad\quad uid = S.runTimeResolver(cs) \land \\
\qquad\quad inst = S.instantiator(uid, ps) \bullet \\
\qquad\quad newInstants(iid) = (inst, uid)) \bullet \\
\qquad openComponents(S, specs) = S'
\end{array}
$$

$$
\begin{array}{|l}
\hline
presentComponent \\
\hline
\Delta SESSION \\
spec? : SPECIFIER \\
presentSpec? : PRESENT\_SPEC \\
\hline
\theta SESSION' = \\
\quad openComponents(\theta SESSION, \{(spec?, presentSpec?)\}) \\
\hline
\end{array}
$$

We can also follow a link from a given link marker in a given instantiation and present all the components for which the associated link(s) has(have) specifiers with a "TO" direction. There may be more than one link involved because there may be more than one link associated with a particular anchor.

followLink

$$\begin{array}{|l}
\Delta SESSION \\
iid? : IID \\
linkMarker? : LINK\_MARKER \\
\hline
\exists\, aid : ANCHOR\_ID;\ links : \mathbb{F}\, LinkComp; \\
\qquad specs : \mathbb{F}(SPECIFIER \times PRESENT\_SPEC) \mid \\
\quad aid = (first(instants(iid?))).linkAnchor(linkMarker?) \wedge \\
\quad links = H.accessor(\!| LinksToAnchor(H, \\
\qquad\qquad second(instants(iid?)), aid) |\!) \wedge \\
\quad first(\!| specs |\!) = \{ s : SPECIFIER \mid \exists\, linkc : LinkComp \mid \\
\qquad linkc \in links \bullet s \in \mathrm{ran}(link^{\sim}(base(linkc))).specifiers \} \wedge \\
\quad (\forall\, s : specs \bullet (first(s)).direction = TO \wedge \\
\qquad\quad second(s) = (first(s)).presentSpec) \bullet \\
\qquad \theta SESSION' = \\
\qquad\qquad openComponents(\theta SESSION, specs)
\end{array}$$

Opening up a new instantiation on a newly created component is modeled by the *newComponent* schema.

newComponent

$$\begin{array}{|l}
\Delta SESSION \\
component : COMPONENT \\
baseComp? : BASE\_COMPONENT \\
ps? : PRESENT\_SPEC \\
presentSpec? : PRESENT\_SPEC \\
\hline
history' = history \frown \langle CREATE \rangle \\
(H', component) = CreateNewComponent(H, baseComp?, ps?) \\
\exists\, uid : UID;\ inst : INSTANTIATION;\ iid : IID \mid \\
\quad iid \notin \mathrm{dom}\ instants \bullet \\
\quad inst = instantiator(uid, presentSpec?) \wedge \\
\quad uid = H'.accessor^{\sim}(component) \wedge \\
\quad instants' = instants \oplus \{ iid \mapsto (inst, uid) \}
\end{array}$$

The schema *unPresent* models the removal of an instantiation.

unPresent

$$\begin{array}{|l}
\Delta SESSION \\
iid? : IID \\
\hline
H' = H \\
history' = history \frown \langle UNPRESENT \rangle \\
instants' = \{iid?\} \ntriangleleft instants
\end{array}$$

Instantiations can be modified by editing them. Editing an instantiation does not cause a change in its corresponding component. An explicit save operation is required to save the result of an edit (or many edits).

editInstantiation

$$\begin{array}{|l}
\Delta SESSION \\
instantiation? : INSTANTIATION \\
iid? : IID \\
\hline
H' = H \\
history' = history \frown \langle EDIT \rangle \\
iid? \in \mathrm{dom}\ instants \\
instants' = instants \oplus \\
\qquad \{iid? \mapsto (instantiation?, second(instants(iid?)))\}
\end{array}$$

realizeEdits

$$\begin{array}{|l}
\Delta SESSION \\
iid? : IID \\
\hline
history' = history \frown \langle SAVE \rangle \\
instants' = instants \\
\exists\, c : COMPONENT;\ inst : INSTANTIATION;\ uid : UID \mid \\
\qquad\qquad inst = first(instants(iid?)) \wedge \\
\qquad\qquad uid = second(instants(iid?)) \wedge \\
\qquad\qquad c = realizer(inst) \bullet \\
\qquad H' = ModifyComponent(H, uid, c)
\end{array}$$

To be complete we must allow a component to be deleted. Since a component is identified by its instantiation, the component to be deleted must have been instantiated. We also must remove any other instantiations for that component.

deleteComponent

$\Delta SESSION$
$iid? : IID$

$history' = history \frown \langle DELETE \rangle$
$iid? \in \mathrm{dom}\ instants$
$\exists\, uid : UID \mid uid = second(instants(iid?)) \bullet$
$\quad H' = DeleteComponent(H, uid) \land$
$\quad instants' = \{iid?\} \ntriangleleft instants$

A session finlly ends when it is closed out. Notice that the default is not to save the results of any changes to instantiations.

closeSession

$\Delta SESSION$

$H' = H$
$history' = history \frown \langle CLOSE \rangle$
$instants' = \varnothing$

We can model a read-only SESSION with the following schema:

READ_ONLY_SESSION

$SESSION$

$\{SAVE, CREATE, DELETE\} \cap \mathrm{ran}\ history = \varnothing$

4 Conformance with the Reference Model

One reason to have a reference model for hypertext is to try to answer the ascertain whether a purported hypertext system actually warrants being called a hypertext system. So, given an actual hypertext system how do we show that it meets, or is conformant with the model? The best guidance for answering this question comes from the VDM experience under the heading of *data reification* as described, for example, in Chapter 8 of Cliff Jones' book [13] on software development using VDM. First, we must exhibit total functions, called *retrieve functions* which map the actual types and functions from given (actual) hypertext system to each of the following types and functions of the model. We must also demonstrate *adequacy* – that there is at least one actual representation for each abstract value. Obviously, the retrieve functions must satisfy the invariants which are given for the data types and functions. An informal way of saying this is that everything which is expressible or realizable in the model must be expressible or realizable in the actual system.

In actuality our model is much more powerful than necessary. In particular

- By admitting multi-way links and links to links in the model, we put a fairly heavy burden on any implementation.
- Many hypertext systems do not have the notion of composites.
- Some hypertext systems, such as KMS, do have not have links with both an explicit source and destination. Thus requiring discrimination amongst all the values of type *DIRECTION* is too much.

We are currently working on a "minimal" model which address the above items and others as may be necessary.

The following list summarizes the given sets (base types), abstract types, functions, and operations which must have actual realizations in a hypertext system conforming to the model.

1. GivenSets.

   UID
   COMPONENT_SPEC
   PRESENT_SPEC
   ANCHOR_ID
   ANCHOR_VALUE
   ATOM
   ATTRIBUTE
   VALUE
   IID
   BASE_INSTANTIATION
   LINK_MARKER

2. Abstract types.

DIRECTION
ANCHOR
SPECIFIER
LINK
COMP_INFO
BASE_COMPONENT
COMPONENT
HYPERTEXT
INSTANTIATION
OPERATION
SESSION

3. Storage layer functions.

CreateNewComponent
DeleteComponent
ModifyComponent
AttributeValue
SetAttributeValue
AllAttributes
LinksToAnchor

4. Runtime layer operations (schemas).

openSession
presentComponent
followLink
newComponent
unPresent
editInstantiation
realizeEdits
deleteComponent
closeSession

5 Concluding Remarks

Development of the Dexter model is still in its very early stages. As discussed in Section 4, the model as currently stated is far more powerful than any existing hypertext system. The provisions for n-ary links and for composite nodes, for example, are intended to accomodate the design of future hypertext systems. No existing system that we have examined includes both n-ary links and composite nodes. The result is that no existing system 'conforms to' the model in the sense that it supports all of the mechanisms that the model supports. The solution to this problem is to make some mechanisms 'optional', resulting in a family of interrelated models that support differing sets of optional mechanisms. The weakest model, for example, would have no composites and only binary links. The strongest model would be the Dexter model in the present form. Conformance to the model could then be conditionalized on the exact set of mechanisms supported. Systems would be compared on the basis of the set of mechanisms that they do support.

A related issue involves a number of consistency restrictions that the present model imposes. For example, when creating a link the model requires that all of its specifiers resolve to existing components. This restriction prevents the creation of links that are 'dangling' from the outset. The model does not, however, include any restrictions that prevent the creation of dangling links via the deletion of linked-to components. This restriction adequately represents the consistency guarantee of KMS. But its is overly restrictive for Augment, which allows creation of initially dangling links. In contrast, its is not restrictive enough for NoteCards and HAM which prevent dangling links at all times. As in the case of mechanisms, restrictions of this sort will have to be made optional in the model. Conformance to the model can then be conditionalized on appropriate choices of restrictions. As in the case for mechanisms, systems can compared on the basis of the set of restrictions that they enforce.

The model has yet to be compared in detail to the hypertext systems it is designed to represent. Clearly, a necessary step in the development of the model is to formally specify (in Z) the architecture and operation of a number of 'reference' hypertext systems using the constructs from the Dexter model. These reference systems should be chosen to represent a broad spectrum of designs, intended application domains, implementation platforms, etc. This enterprise would provide valuable feedback regarding the adequacy and completeness of the model. In particular, it will help asess whether the model provides sufficient mechanisms for representing the

```
<hypertext>
    <component>
        <type> text </type>
        <uid> 21 </uid>
        <data> This is some text .... </data>
        <anchor>
            <id> 1 </id>
            <location> 13 </location>
        </anchor>
    </component>
    <component>
        <type> text </type>
        <uid> 777 </uid>
        <data> This is some other text .... </data>
        <anchor>
            <id> 1 </id>
            <location> 13-19 </location>
        </anchor>
    </component>
    <component>
        <type> link </type>
        <uid> 881 </uid>
        <specifier>
            <component_uid> 21 </component_uid>
            <anchor_id> 1 </anchor_id>
            <direction> FROM </direction>
        <\specifier>
        <specifier>
            <component_uid> 777 </component_uid>
            <anchor_id> 1 </anchor_id>
            <direction> TO </direction>
        <\specifier>
    </component>
</hypertext>
```

Figure 5: Example of a trivial interchange format derived from the model.

important (common) abstractions found in the reference systems. It will also provide feedback on the 'naturalness' of the model, i.e., on whether the specification of the reference systems in Dexter terms feels 'natural' or whether the abstractions found in certain systems must be excessively massaged to fit into the Dexter abstractions.

Despite its early stages of development, the model has already been useful in developing hypertext interchange standards. As described in the panel on interchanging hypertexts at the Hypertext 89 Conference [16], a number of efforts have been started to operationalize the abstractions of the Dexter model in the form of interchange formats. Figure 5 shows an example of one such format. This format was used for experimenting with the interchange of hypertexts between NoteCards and Hypercard. As can be seen from the figure, the format is a fairly straightforward rendering of the entities found in the Dexter model into a SGMLish syntax. This format is by no means a well-developed interchange standard. But it does suggest that the Dexter model provides a good basis from which to develop such standards. In fact, because the model is an attempt to provide a well-defined and comprehensive model, it is an ideal basis for developing a comprehensive standard for interchanging hypertexts between widely differing systems.

References

[1] Akscyn, R., McCracken, D.L., & Yoder, E.A. KMS: A distributed hypertext for managing knowledge in organizations. *Communications of the ACM*, 31(7), 1988, 820-835.

[2] Campbell, B. & Goodman, J.M. HAM: A general purpose hypertext abstract machine. *Communications of the ACM*, 31(7), 1988, 856-861.

[3] Conklin, J. Hypertext: A survey and introduction. *IEEE Computer*, 20(9), 1987, 17-41.

[4] Delisle, N. & Schwartz, M. Neptune: a hypertext system for CAD applications. *Proceedings of ACM SIGMOD '86*, Washington, D.C., May 28-30, 1986, 132-142.

[5] Englebart, D.C. Authorship provisions in Augment. *Proceedings of the IEEE COMPCON*, Spring, 1984, 465-472.

[6] Englebart, D.C. Collaboration support provisions in Augment. *OAC Digest: Proceedings of the 1984 AFIPS Office Automation Conference*, Los Angeles, February 20-22, 1984, 51-58.

[7] Feiner, S., Nagy, S., & van Dam, A. An experimental system for creating and presenting interactive graphical documents. *ACM Transactions on Graphics*, 1(1), 1982, 59-77.

[8] Goodman, D. *The Complete HyperCard Handbook*. New York: Bantam Books, 1987.

[9] Halasz, F.G., Moran, T.P., & Trigg, R.H. NoteCards in a nutshell. *Proceedings of the 1987 ACM Conference of Human Factors in Computer Systems (CHI+GI '87)*, Toronto, Ontario, April 5-9, 1987, 45-52.

[10] Halasz, F.G. Reflections on NoteCards: Seven issues for the next generation of hypermedia systems. *Communications of the ACM*, 31(7), 1988, 836-855.

[11] *Proceedings of Hypertext 87*, Chapel Hill, NC, November 13-15, 1987. Available from ACM Press, order number 608892.

[12] *Proceedings of Hypertext 89*, Pittsburgh, PA, November 5-8, 1989. Available from ACM Press, order number 608891.

[13] Jones, C.B. *Systematic Software Development Using VDM.* Prentice-Hall International, Hertfordshire, England, 1986.

[14] Lange, D.B. A formal approach to hypertext using post-prototype formal specification. Dept. of Computing Science, Technical University of Denmark, Oct. 31, 1989.

[15] Meyrowitz, N. Intermedia: The architecture and construction of an object-oriented hypermedia system and applications framework. *Proceedings of the Conference on Object-oriented Programming Systems, Languages, and Applications (OOPSLA '86)*, Portland, OR, September 29 - October 2, 1986, 186-201.

[16] Oren, T. Panel: Interchanging hypertexts. *Proceedings of Hypertext 89*, Pittsburgh, PA, November 5-8, 1989, 379-381.

[17] Pearl, A. Sun's Link Service: A protocol for open linking. *Proceedings of Hypertext 89*, Pittsburgh, PA, November 5-8, 1989, 137-146.

[18] Shneiderman, B. *Hypertext on Hypertext.* Addison-Wesley: New York, 1989.

[19] Spivey, J.M. *The Z Notation.* Prentice-Hall International, Hertfordshire, England, 1989.

[20] Trigg, R.H. & Weiser, M. TEXTNET: A network-based approach to text handling. *ACM Transactions on Office Information Systems*, 4(1), 1986, 1-23.

[21] Walker, J. Document Examiner: Delivery interface to hypertext documents. *Proceedings of Hypertext 87*, Chapel Hill, NC, November 13-15, 1987, 307-323.

[22] Walker, J. Supporting document development with Concordia. *IEEE Computer*, 21(1), 1988, 41-59.

[23] Yankelovich, N., Haan, B., Meyrowitz, N., & Drucker, S. Intermedia: The concept and the construction of a seamless information environment. *IEEE Computer*, 21(1), 1988, 81-96.

STANDARDIZATION OF HYPERMEDIA: WHAT'S THE POINT?

A Position Paper

Hypertext Standardization Workshop

National Institute of Standards and Technology
National Computer Systems Laboratory
January 16-18, 1990

Shoshana L. Hardt-Kornacki, Louis M. Gomez, John F. Patterson
Bellcore
445 South Street
Morristown, NJ 07960-1910
(201) 829-4528 shoshi@bellcore.com

Abstract

In this paper we present multiple views on the issue of standardization of Hypermedia systems that operate over a global heterogeneous information network. To aid our analysis we introduce a reference model that captures the information flow and the information control aspects from the viewpoint of the user. This model is then used to focus the analysis of Hypermedia systems from a variety of perspectives, such as overall resources, network communication, interface building, and application writing. Based on our analysis we conclude that at this time, the components of Hypermedia systems that are ready for standardization are not necessarily Hypermedia-specific. Moreover, we strongly believe that the Hypermedia-specific aspects of these systems are not yet ready for standardization and we question the wisdom of ever standardizing certain Hypermedia specific components such as the user interface or the navigation tools. In addition, we conjecture that it may be desirable to standardize a generic set of tools that can be used to build these components so as to guarantee that the access to the information stored in future Hypermedia systems will not be impaired.

ANGLES ON STANDARDIZATION

Intrinsic to the quest for standardization is the desire to make artifacts designed by different people in different places at different times compatible in relation to some predefined tasks. If we ask why one should attempt to standardize HyperText and Hypermedia technologies, we should look for the answer in efforts to combine pieces of information, text, graphics, still images, audio, video, animation and the like, which were created by different people in different places at different times. From this perspective it follows that it is reasonable to consider such standardization efforts only if we are willing to view the system as operating on a very large heterogeneous network.

Multimedia is a very complex artifact. It requires large amounts of resources and human involvement. Because of its potential as a new medium in which the human can seek, express and control knowledge, human interface considerations are of crucial importance. Much of the complexity involved in running the support hardware and software that make Hypermedia systems a reality must remain hidden from the human and should proceed automatically. This implies the smooth and efficient transfer of information and control between many machines, each with its own capabilities for communication and information handling. Furthermore, it implies that the overall speed of the composite system should remain mostly unaffected by the global configuration of the various information sources and conduits to enable synchronization.

The standardization of an artifact as complex as Hypermedia involves the standardization, or at least a thorough understanding, of the evolutionary trends existing today in the Hypermedia supporting technologies. Any attempts to freeze a version of a rapidly evolving system should be carefully engineered so as to guarantee uninterrupted progress. Therefore, one of the more important challenges is to decide which aspects of Hypermedia need to become a standard and which aspects are better off left alone. This decision should be based on a model of the functionality of the system, a model flexible enough to allow unexpected technological developments. To illustrate this point let us consider two extreme scenarios for Hypermedia functionality. In the first scenario a single user is running a standalone application on a workstation. In the second scenario a user is running a shared application, which includes real-time communication via broadband networks with other users and with a variety of information gateways to distributed data sources. Undoubtedly, the complexity of the issue of standardization and its implications on information sharing are of different proportions in the two scenarios. In the first case, standardization must guarantee the compatibility of applications in many present and future environments. In the second case, standardization will guarantee complete information sharing across authors, users and machines. It is the second scenario which can benefit the most from standardization and at the same time is in the most fragile developmental phase and hence requires special handling.

There are at least three reasons to embark on standards efforts. First it may be valuable to come to some agreement on a Hypermedia independent environment which will support this brand of computation. Second, standards may focus on the representation of data objects use in Hypermedia applications. And third, a standards effort might concentrate its energy providing a standard human interface for applications that are browsing and information retrieval intensive.

With respect to the first point, a standard reference model which supports Hypermedia almost certainly shares many, if not all, its attributes with reference models for most other applications. It may be useful, however, for Hypermedia practitioners to determine where, in a layered reference model Hypermedia applications exert most of their impact. Later in this paper we outline a general reference model to facilitate discussion of this sort.

Hypermedia applications are intimately concerned with data objects of various types and their interrelation. Because of their complex linking structure and multiple media flavor, Hypermedia applications, in all likelihood, require that data objects have detailed and explicit representations. Rich and flexible standard representations will be of great value to Hypermedia implementers in matters of exchange and authoring. It is also the case, however, that these very same objects (e.g. image, video) and their underlying representations are also critical to many other classes of applications where exchange is important but has nothing to do with Hypermedia. Therefore, we question the prudence of Hypermedia-based object presentation standards. It would seem that Hypermedia practitioners should, again, consider the unique impact that hypertext applications might have on current and emerging object presentation standards efforts. We offer some conjectures in this regard in the context of a reference model.

While the defining characteristic of Hypermedia is its linking structure, its most often cited benefit is as an aid to human intellect. It may be reasonable then, for Hypermedia practitioners to look for standards in the human interface to realize this cognitive benefit. We conjecture that this route is at best premature and at worst naive. A standard Hypermedia human interface is premature simply because there does not exist very much solid information about the sorts of Hypermedia design features that people find helpful. This state of affair makes it virtually impossible to code high level standards which could sensibly and practically apply to the multiplicity of potential Hypermedia applications. Readiness aside, such a standards quest may not be prudent. The target domain of an application often changes fundamental qualities of its interface. Given the complexity of Hypermedia application domains, it may be more prudent to build highly stereotype applications optimized for the communication and problem solving needs of a particular domain rather than a vanilla consistent interface that does not accommodate the rich variation in Hypermedia applications.

In this paper, we center our discussion around a view of the Hypermedia system from the user's perspective. If we follow the information and control as they flow from the user's terminal to the actual database, we cross at least eight functional levels. These levels are described in the next section, followed by an illustration of their descriptive power in two examples of prototype Multimedia systems. This illustration is followed by a discussion of the Hypermedia system from other perspectives and the implications of this decomposition into levels on standardization.

A REFERENCE MODEL FROM THE USER'S VIEWPOINT

Like many other dynamic systems with a high degree of complexity, Hypermedia can be viewed from multiple perspectives. Each perspective reveals a dimension along which hierarchical description levels can be stacked and interdependencies between structure and function revealed.

Level 6	**File System**	**Virtual InterProcesses Communication Mechanism**	**Broadband Network**
Level 5	**Virtual File System**		
Level 4	**Presentation Objects**		
Level 3	**Dialogue/Applications**		
Level 2	**Virtual Terminal**		
Level 1	**Actual Terminal**		

Figure 1
Six Plus Two Level Reference Model Describing the Passage of Information and Control From the User at the Actual Terminal to the Actual Information Source. We View Level 3 and 4 as the Only Hypermedia Specific Levels.

Imagine the way a Hypermedia system looks from the perspective of the user. From this perspective, both information and control are conveyed through layers of interpretation until they reach their destination which, in this case, is an arbitrary collection of actual file systems created by arbitrary authors and located at remote sites which may be unknown to the user. We chose to separate the path of information and control into eight independent layers, each with its own set of primitive operations and data elements. Consequently, implicit to the construction of this reference model is the assumption that the functionality of the overall system is decomposable. However, keep in mind that many complex artifacts are only nearly decomposable, namely, their actual implementation involves "mixing" of levels due to strong pragmatic considerations. Therefore, we consider this model an idealization which serves as a general guideline during system design and evaluation.

In Figure 1 we introduce the eight level model and represent it as a "six plus two" level model. This is because, the virtual interprocess communication mechanism and its actual network implementation can be involved in the information transmission process anywhere along the path between the actual terminal and the actual file system and hence could not be placed in any particular location on the stack.

Undoubtedly, the reference model, at the level of detail shown in Figure 1, may describe any interactive distributed computer system. This raises the question of where do we perceive the Hypermedia specific components of the system to reside. In attempting to answer this question one may realize that any computer system, when examined very closely, exhibit many of what one may consider at least Hypertext specific characteristics. For example, the Unix® file system provides much of the functionality of a Hypertext system, without, perhaps, a sylized user interface. We will return to this point shortly, after we briefly review the levels shown in Figure 1.

The bottom two levels in Figure 1 describe the terminal and the virtual terminal. Like all virtual devices, the virtual terminal provides a level of description that is implementation independent. The primitive operations comprising the virtual device description are implemented in every device to the best of that device's actual capabilities. Like all virtual devices, it represents an additional level of processing of information, which is the price one must pay for flexibility. With the virtual terminal level of description, dialogues (applications) can be constructed (level 3) that are implementable on the virtual terminal and which have as primitive operations user interaction activities. The dialogue level is the "information browsing" level and the value of separating it from the virtual terminal level is that it enables the application writer to tailor the interface to the applications and to the targeted user community in a terminal independent fashion. The level of description of the Presentation Objects (level 4) contains packets of information stored in a form that can be displayed by any interface. The database containing these objects is represented in level 5. Notice that operations at each level in the stack except the top three are represented in terms of primitive operations of the level below it. In the case of the top three levels, which are separated in Figure 1 by a double line, the order is reversed. This is because the presentation objects are implemented in terms of the virtual file system, and the virtual file system is implemented in terms of the actual file system. This reversal property is an essential part of any description scheme that, similar to our scheme, follows the path of information and control between the user and some real data -- the scheme has to start with a real object, namely the terminal, and end with a concrete implementation of data. We will not to elaborate on the actual implementation levels of the file system.

Which of the above levels are part of the Hypermedia application and which levels describe the environment? In our work we view the Presentation Objects and the interface (levels 3 and 4) as part of Hypermedia and they will be discussed in more details in the next section. We view the other description levels as representing the supporting infrastructure for global Hypermedia systems and for most other applications. Currently, this supporting infrastructure is not standardized, e.g., the virtual terminal and the virtual file system are not standards, and broadband communication networks are far from standardized. Given this view, one may question, as we did in the first section, the wisdom of standardizing Presentation Objects and aspects of interfaces before, at least, stable sketchs of a standard virtual terminal and a standard virtual file system are agreed upon.

In the next section we will examine standardization issues from various viewpoint, but before doing so we illustrate the value of the reference model presented in Figure 1 in two examples. To demonstrate how the reference model provides structure to the functionality of Hypermedia systems, we look at the following two systems from the domain of Customized Electronic Information Delivery. Customized Electronic Information Delivery systems provide users with variable information streams. Regarding the level of editing of the information items delivered by such systems we can imagine two extremes -- highly stylized, long, magazine like, articles, and short raw articles directly from the news wires. The Electronic Magazine (Judd and Cruz, 1989) is an example of the former, and the Passive Information Grazing system (Bussey *et al*, 1989) is an example of the latter.

The Electronic Magazine research prototype displays multimedia articles through a stylized user interface providing the user with navigation and orientation tools. In addition, the magazine contains multimedia authoring tools and a mark-up language. Figure 2 presents a glance at the Electronic Magazine from the perspective of the reference model presented above.

Actual Terminal	Sun-3 Color Monitor
Virtual Terminal	SunView™ Window System
Dialogue/Applications	Multimedia Interface Navigation tools
Presentation Objects	Stylized Multimedia Articles SGML Based Mark-Up Language Authoring tools
Virtual File System	Linked Database of Multimedia Articles
Actual File System	Unix® Files
Virtual InterProcess Communication Mechanism	None
Actual Network	None

Figure 2
Description of the Electronic Magazine Prototype

SunView is a trademark of Sun Microsystems, Inc.
Unix is a registered trademark of AT&T.

Actual Terminal	Sun-3 Color Monitor
Virtual Terminal	X Window System™
Dialogue/Applications	Simple Divided Screen Navigation Tools
Presentation Objects	Unedited Multimedia News Items
Virtual File System	Categorized Articles
Actual File System	The Oracle Database
Virtual InterProcess Communication Mechanism	None
Actual Network	EXPANSE (see Bussey *et al 1989).*

Figure 3
Description of the Passive Information Grazing Prototype

The research prototype of the Passive Information Grazing System provides the user with a continuous stream of multimedia information through a simple interface. Before reaching the user the information passes through a filter eliminating articles that according to a personalized user profile, are of no interest to the user. Figure 3 shows a brief overview of the system from the perspective of the reference model.

INTERSECTING DIMENSIONS AND STANDARDIZATION ISSUES.

Hypermedia systems require a very rich infrastructure. Even though they may be viewed as mere application programs, they put a severe strain on existing computational and communication resources. They push today's technologies to their limits. Therefore, when it comes to standardization it may be ill advised to consider Hypermedia as a standalone application and not as a system that is closely coupled with the development of its infrastructure. For example, from the viewpoint of resources, the actual performance and capabilities of the system are affected by resources available at each of the levels described in Figure 1. Parameters such as network reliability and speed, information storage capacity, CPU "horse power", and terminal capabilities

---

X Window System is a trademark of MIT.

may play a major role in defining the future shape of Hypermedia applications.

Keeping the Hypermedia dependencies on its infrastructure in mind, we will proceed to discuss Hypermedia and its standardization from the view point of the Hypermedia application writer. According to the reference model presented in Figure 1, the application writer is equipped with terminal independent and file system independent authoring tools. In our framework, the application writer is responsible for producing the Presentation Objects, and the User Interface. The Presentation Objects are the key elements of the system. A collection of them resides in the virtual file system, and they are displayed on the interface. Aspects of their structure are given in Figure 4.

Object Description:
- links
- attributes
- authorization
- displaying methods

Object Presentation:
- envelope
- body

Figure 4
The Structure of Presentation Objects.

It is important to note that in the context of the current discussion, the Presentation Objects provide a way to carve-up meaningful presentable pieces of multimedia information. This is due to the fact that the Presentation Objects contain sufficient specification to guarantee that they can be displayed, classified, stored, retrieved, and filtered in a global Hypermedia system. Also, they essentially represent an "Object Oriented Approach" to Hypermedia information representation and management.

We view Presentation Objects as consisting of two main parts -- the Description part and the Presentation part. The Description part contains the links that the object has to other objects, attribute of the object such as its size and the resources it needs, information about authorization and authoring tools, and methods to display it. The Presentation part of the object contains the envelope and the body. The envelope contains preview information about the body of the object, e.g. title, abstract, video clip etc. The body is (a pointer to) the content of the object.

The level of the dialogue captures user interface and session management issues. Some of its functionality is given in Figure 5.

Current Status

Available Objects

Open Objects

Navigation Tools:
  within object navigation
  between objects navigation

Authoring Tools

Displaying Tools

Figure 5
The Level of the Dialogue (Applications)

CONCLUSIONS

We are now in a position to consider our central problem here: What do we need to standardize in order to guarantee information sharing in Hypermedia systems that operate over a global heterogeneous information network?

The standardization of the virtual terminal, the virtual file system, and the virtual interprocesses communication mechanism should come first. These standards will guarantee that any application can run on the standard virtual terminal irrespective of the terminal and the actual file system used, and that any network can be used for communication given that it can emulate the virtual network. Regarding the Hypermedia components, the Presentation Objects should be the next in line for standardization. However, as stated in the opening section, since at the present time we still cannot assess the potential multimedia capabilities of the future we must wait for the above standards before we consider freezing the form of the Presentation Objects and their database.

If we now look at the situation where all the levels in Figure 1 are a standard except the application level we immediately realize that there is no point in standardizing it. The fact that the levels above and below it are a standard impose a strong enough constraint that produces a standard set of tools to build the software at that level. This approach sets the functionality of Hypermedia but not its "look and feel". We believe that at this point it is still inappropriate to standardize "look and feel" of Hypermedia because not enough is known about the relationship between the users' cognitive skills and personal preferences and the benefits that Hypermedia has to offer to them. Therefore, at this point, a standard user interface may defeat the purpose of user-friendliness and may make personalized access to information impossible.

BIBLIOGRAPHY

Bussey H., Edigo C. Kaplan A. Rohall S. and Yuan R. (1989). *Service Architecture, Prototype Description, and Network Implications of a Personalized Information Grazing Service*. Submitted to Infocom '90.

Judd T.H. and Cruz G.C (1989). *Customized Electronic Magazines - Electronic Publishing for Information Grazing*. Advanced Printing of Paper Summaries, Electronic Imaging '89, Vol. 1, pp. 504 - 509.

A Formal Model of Hypertext[*]

Danny B. Lange

Brüel & Kjær Industri A/S[†]
DK - 2850 Nærum, Denmark
Tel: +45 42 80 05 00
email: danny.lange@bk.dk

Department of Computing Science[‡]
Technical University of Denmark
DK - 2800 Lyngby, Denmark

January 22, 1990

Abstract

In this paper a formal specification of an abstract model of hypertext is presented. The Vienna Development Method (VDM) is used in this specification. Experiences with a prototype hypertext system and studies of other existing hypertext systems are captured in this formal specification. Basically datamodel of hypertext is suggested. In this model three main abstract data types of hypertext are formally defined: nodes, networks and structures. The abstract data types are applied to the concepts of object-oriented databases and a "hyperbase" is defined.

1 Introduction

Hypertext is becoming a well-known technique for information representation and management. Different research projects show that hypertext has many potential applications that are just beginning to be explored: textbooks, dictionaries, encyclopedias and software engineering [Hypertext 1989]. At the Hypertext'89 Conference a wide range of hypertext products were presented. They all covered many different aspects of hypertext. But, they had one thing in common. When it comes to means of interchange and communication between these systems they are all doomed to fail.

In this jungle of different systems, publishers of hypertexts must worry about portability of their works between different hypertext systems to ensure that they don't depend to much upon the success of one system. The users of hypertext systems must worry about the supply of hypertexts or use of hypertext organization of long-lived project documentation stored in a specific hypertext system, making the data inaccessible for other (hypertext) systems.

Steps toward interchange and communication between open hypertext systems must be based on formal and abstract models of hypertext to which all existing and hopefully future systems can be related. In the last few years an increasing number of papers on hypertext and its application has been published. Only a very small part of this work has been concerned with the formal treatment of hypertext. There is clearly a need for a more formal approach to hypertext since one can claim that hypertext is driven by user interface and implementation considerations [Halasz & Conklin 1989]. Looking through the Hypertext'89 Proceedings [Hypertext 1989] one will find dissapointing few papers on the more formal and abstract aspects of hypertext. However, attempts to present more formal models of hypertext have appeared [Delisle & Schwartz 1987] [Garg 1988] [Stotts & Furuta 1989] [Consens & Mendelzon 1989]. This paper presents a formal model of hypertext, using the Vienna Development Method (VDM) [Bjørner & Jones 1982] [Jones 1986]. VDM supports the top-down development

---

[*] A version of this paper emphasizing a formal specification methodology and with different technical details, but inevitably overlapping in the datamodel facet with the present paper, is being presented at VDM'90 and published in the conference proceedings by kind permission of the Programme Comittee and the editors.

[†] Author's Present Address

[‡] A part of the work has taken place at the Technical University of Denmark

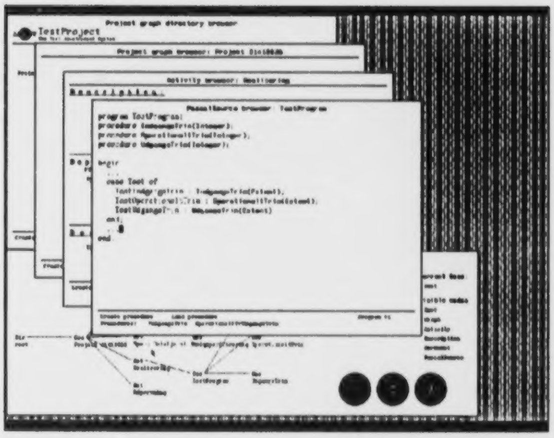


Figure 1: A Snapshot of the Prototype

of software systems specified in a notion suitable for formal verifikation. The specifications are based on a datamodel using high-level types as set, list, map and cartesian products. Function specification are written in predicate logic, using pre-conditions stating the properties that the inputs must satisfy, and post-conditions which states the relationship of inputs to outputs.

At Brüel & Kjær[1] we have developed a prototype of a hypertext system. The prototype was developed on a SUN3 workstation[2] using an expert system shell called ART[3]. The prototype was written partly in ART's rule-based language and COMMON LISP [Steele 1984] using a window based user interface, see figure 1. The prototype has fulfilled several aims. First it has given the developers a feeling of what hypertext is all about, by working with the prototype. Secondly the ideas of hypertext has easily been communicated to non-experts and potential users.

Our experiences with this prototype and studies of hypertext systems as Hypercard, Hyperties, Neptune, KMS, Nodecards, etc. is captured in the formal specification presented in section 2 and section 3 in this paper. In section 2 the datamodel of hypertext is presented by domain equations giving a formal definition of the primitives of hypertext, introducing the three main concepts: nodes, links and structures. In section 3 the datamodel is extended with a set of operations in an object-oriented way, defining abstract datatypes of nodes, links and structures. Our experiences with this formal model and future work are discussed and concluded in section 4. Detailed pre-/post- specifications of the specified operations can be found in appendix A.

2 Developing a Basic Datamodel of Hypertext

The hypertext datamodel has evolved on basis of the experience with our prototype and our general knowledge to the domain. The model will include the concept of nodes and their interior, links between nodes and between fields and buttons inside the nodes. Different kinds of links are described: N-ary links, second order links and active links. Additionally the idea of having structures organizing nodes in e.g. hierarchies, is introduced.

In the following a datamodel of hypertext is developed through stepwise refinement. Initially the meaning of hypertext is defined as a database that has active cross-references, allowing the user to have nonsequential access to a text thereby making the reading process nonlinear. A hypertext can be modelled as a set of nodes and a collection of links where the nodes are documents and the links are cross-references.

---

[1] Brüel & Kjær Industri is a company that designs and manufactures high-precision electronic measuring instruments.

[2] Sun Workstation is a registered trademark of Sun Microsystems, Inc.

[3] ART (Automated Reasoning Tool) is a registered trademark of Inference Corporation.

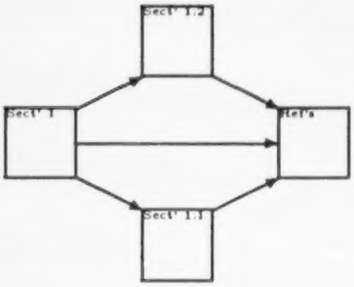


Figure 2: Example of linked nodes

1.0 *Hypertext* :: *Nodes* × *Links*
2.0 *Nodes* = *"chunks of information"*
3.0 *Links* = *"cross-references"*

2.1 Nodes - Units of Information

An information fragment in a hypertext is called a node. Thus, hypertext is made up of a collection of distinct named information fragments. Conceptually this information fragment usually describes a single concept or topic. The names may be assigned explicitly by the user or they can be assigned automatically.

In some hypertexts it might be necessary to divide the nodes into several different types: document, illustration, annotation, etc. Thus, it must be possible to add attributes and attribute values to nodes.

4.0 $Nodes = Nid \underset{m}{\rightarrow} (Node \times Attributes)$
5.0 *Node* :: *"information"*
6.0 *Nid* :: TOKEN

2.2 Links - the Glue that Holds Hypertext Together

A connection between two nodes is called a link. When a link is activated, say by a mouse click, one can jump to the node the link points to. A hypertext network is made up of a collection of uniquely named links. Links can be used to transfer the reader to an new topic, provide access to an annotation or footnote, show a reference and so on. Conceptually a link is directed, i.e. it points from one node to another, having an origin called the anchor and an end point called the destination. However this does not mean that links are unidirectional, that is, the passage is not only one-way. One can always pose the question: who points to me?

In figure 2 one can see an example of a document consisting of a section, two subsections and a reference list. The section is connected to its subsections through node to node links. All three items link to a common reference list. The section node might contain the text of the introduction to the two subsections, and the nodes of the subsections, contains the text of the subsections. Below the concept of linking is restricted to only concern connections between entire nodes. In section 2.4 the model is extended to include links between the contents of one node and another node.

A hypertext system may have only one type of link or it may have several types. The link type can reflect the type of information it is pointing to, making it possible for the user only to view links of a certain type. Different types of links in a document could be references to related articles or reviewers annotations. To represent this variety of linktypes, they can be attributed in the same manner as for nodes.

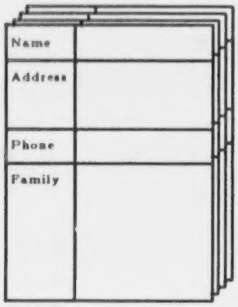


Figure 3: Example of the use of schema

7.0 $Links = Lid \xrightarrow{m} Link$
8.0 $Link :: Connections \times Attributes$
9.0 $Connections :: Anchor \times Destination$
10.0 $Anchor, Destination = Nid$
11.0 $Lid :: \mathrm{TOKEN}$

An important point in hypertext is the support for collaborative work. If several people are reviewing and annotating the same hypertext, they all use the common network made by the author of the document. To this common network each individual can add a personal subnetwork reflecting their own need for referencing across the common network and including references for their annotations. Looking at other persons sub-networks, one can inspect their annotations, possibly realizing that further comments on specific topics are needless, thus saving time in a review process. This does not remove the need for attributed links. One may still need to add individual information to the link, like the time when it was created, why it was created, etc.

12.0 $Networks = Nwid \xrightarrow{m} (Links \times Attributes)$
13.0 $Nwid :: \mathrm{TOKEN}$

2.3 Slots - the Interior of the Node

Conceptually the node can cover a wide range of applications, i.e. representing a chapter or section in a document, function definitions in the source text of programs, organizing information on notecards, etc. Obviously there is a need for a substructure in the interior of the node.

A slot is a kind of template for the contents of the node. It can be compared to the record datatype in programming languages. A node has a collection of unique named slots, each having some kind of textual content. An example of the use of schema in a node is shown in figure 3. In this example information on individuals is organized in an archive. For each person exists one basic "card" carrying a specific set of information: name, address, phone and family. "Cards" can be annotated and one can make references between the "cards". In the family slot, one can mention the spouse and make a link to his/her "card". In our model theser "cards" are equal to the node.

Slots can be connection points for links. As anchors and destinations they are identified by the node in which they are embedded and their name.

14.0 $Node = Slid \xrightarrow{m} Slot$
15.0 $Slot :: String \times Attributes$
16.0 $String :: \mathrm{CHAR}^*$
17.0 $Anchor, Destination = \ldots \mid (Nid \times Slid)$
18.0 $Slid :: \mathrm{TOKEN}$

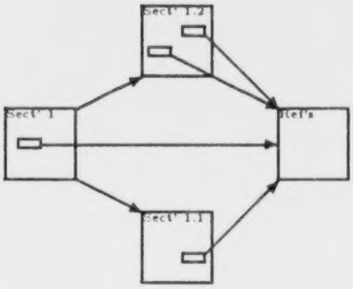


Figure 4: Example of buttons

2.4 Buttons and Fields - the Referential Mechanism

In this section *buttons* and *fields* are introduced. They are the fundamental components of the referential mechanism, one of the most powerful properties of hypertext. Links connecting entire nodes and slots have already been introduced. Now the concept of linking is extended to cover source and destination points inside the nodes. Pragmatically this covers the referential use of links in a hypertext.

A handle is a part of the text inside the slot to which a link can be attached. This makes it possible to establish connections between the contents of one node and another node. A handle is defined as a consecutive sequence of characters in the textual contents of the slot. More precisely by its character position in the text and the span in numbers of characters.

When a link is anchored to a handle, that is, there is an outgoing link from a handel, the text span specified by the handle is called a button. In figure 4 it is shown that one can get from an actual reference in the text to the reference list.

Fields are defined exactly in the same way as the buttons are. We have chosen to distinguish between these two of purely conceptual reasons, thus having fields as one of the possible end-points of links.

The domain of connections is extended to include buttons and fields. From a connections point of view, a button or field is identified by the node and slot in which it is embedded and its handle in that slot.

19.0 *Slot* :: *String* × *Handles* × *Attributes*
20.0 *Handles* = *Hid* $\xrightarrow[m]{}$ *Region*
21.0 *Region* :: *Position* × *Length*
22.0 *Position, Length* :: $\mathbb{N}_0$
23.0 *Anchor* = ... | *Button*
24.0 *Destination* = ... | *Field*
25.0 *Button, Fields* :: *(Nid* × *Slid* × *Hid)*

To continue the example, the use of fields makes it possible to follow a reference not only to the reference list but to a certain entry in this reference list, see figure 5. Depending on the user-interface the entry, i.e. the field, is accentuated.

2.5 More on Links - N-ary Links, 2nd Order Links and Active Links

So far only binary links has been treated. Binary links are characterized by one link anchor and one destination point. They match the concept of navigating in a hypertext very well. That is, if one has an end-point of a link, there is only one way to go, if one choses to follow the link.

For structural reasons it may be more appropriate to consider a more general concept of links. N-ary links have one or more link anchors and one or more destination points. In the model this means that a set of link anchors and destination points are bound to the same link. An example of N-ary links is shown in figure 6. In this example three sections in a document refer to a certain article. Following the links, one might first be directed to an entry in an annotated reference list, for reading an abstract, and then to the article itself. In this way the concept of N-ary links forms the basis of following links in several steps, that is being directed to a short description of the destination before actually arriving there.

26.0 $Connections :: Anchor\text{-}\underline{set} \times Destination\text{-}\underline{set}$

Nodes, slots and fields have been discussed as destination points for links. Links pointing at links, called 2nd order links, can be used to point at a collection of connections. It might reflect that a link itself is of special interest, and that the reader after being guided to the link, can chose to study the anchor or destination of the link. Links are identified as connection points by name of the network in which they are embedded, and their own name.

27.0 $Anchor, Destination = \ldots \mid (Nwid \times Lid)$

Active links are links that have anchors or destinations that are function denotations. That is, instead of having links pointing at fragments of text they contain a function. This function is to be interpreted when one is following the link. This kind of a link can be used to generate a view of the data it is anchored to. That could be the generation of a graphical representation of the data each time one is following the link. A function signature is added to the domain of anchors and destinations. The domains of the arguments and the results of the function are not specified in any further detail.

28.0 $Anchor, Destination = \ldots \mid Argument\text{-}\underline{set} \overset{\sim}{\to} Result\text{-}\underline{set}$
29.0 $Argument, Result \quad = \ldots$

2.6 Structures - the Organizers of Hypertext

The hypertext in figure 2 represents the most simple organisation of a hypertext. This example of a hypertext is a set nodes connected by links. A *hierarchy* of nodes in a hypertext is another primitive example of organising an hypertext. It is a way of organizing information into meaningfull parts e.g. documents into sections and subsections. Figure 7 shows such a hierarchy of sections and subsections in a document. The user is usually free to define information structures in traditionally hypertext systems as they are needed. But, the novice user sometimes may require guidance by the hypertext itself, or one may find *ad hoc* organisation of hypertexts potentially dangerous. The problem can be solved by using *structures*.

Structures should prescribe an organization of nodes and networks. They can conceptually be compared to the domain equations in VDM, introducing sets, sequences, maps and the possibility of recursive definitions, e.g. tree data structures. The structures can form a basis for an algebra for structured hypertext documents [Güting et al.].

The use of the set-structure has already been demonstrated and fits well into card-like hypertexts. The map-structure can extend this unordered collection of cards with a facility of direct access by user defined names. Sequences can be used to express interrelationships between nodes as the sequence in

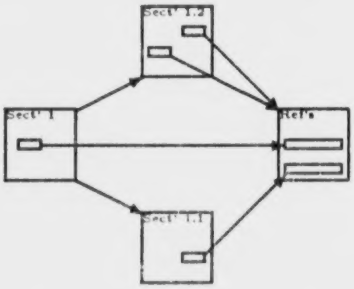


Figure 5: Example of Fields

which they should be visited, e.g. chapters in a book. Defining these structures recursively, makes it possible to make tree structures of nodes.

It should be emphasized that it is not the nodes and networks themselves that are organized in these structures. The structures contains only the names of the nodes and networks. Hence it possible to reuse nodes and networks in several structures. E.g. one can think of a section or figure appearing in more than one book, and thus in several structures.

Structures can be interpretated by filters, to make linear representations of the hypertext, e.g. on paper. A tree structure of a book should intuitively be interpreted by a filter in a top-down left-to-right manner, so that chapter one and the subsections of this chapter are written out before chapter two and so on.

Structures are uniquely identified by their name. Each structure is characterized by having a collection of substructures, each organizing destinations into sets, sequences or maps. The substructures themselves have unique identities and can be destinations, thus making it possible to build more complicated structures. A structure has a root that can identify one of the substructures as being the root of the structure.

30.0 $\mathit{Structures} = \mathit{Sid} \xrightarrow{m} (\mathit{Structure} \times \mathit{Attributes})$
31.0 $\mathit{Structure} = \mathit{Subsid} \xrightarrow{m} \mathit{Substructure}$
32.0 $\mathit{Substructure} = \mathit{Substruc} \times \mathit{Attributes}$
33.0 $\mathit{Substruc} = \mathit{Set} \mid \mathit{Seq} \mid \mathit{Map}$
34.0 $\mathit{Set} = \mathit{Destination}\text{-}\underline{\text{set}}$
35.0 $\mathit{Seq} = \mathit{Destination}^*$
36.0 $\mathit{Map} = \text{TOKEN} \xrightarrow{m} \mathit{Destination}$
37.0 $\mathit{Anchor}, \mathit{Destination} = \ldots \mid \mathit{Sid} \mid (\mathit{Sid} \times \mathit{Subsid})$
38.0 $\mathit{Sid}, \mathit{Subsid} :: \text{TOKEN}$

2.7 The Attributes

Attributes are basically a mapping between names of attributes and their values. The names of the attributes are user defined. The values of the attributes can be of a simple text or numerical type, but one can also expect structured types as known from the attributes of attribute grammars. Among attributes that should be mentioned are version numbers, time for creation, access rights, protection, etc.

39.0 $\mathit{Attributes} = \mathit{Attribute} \xrightarrow{m} \mathit{Value}$
40.0 $\mathit{Attribute} :: \text{TOKEN}$
41.0 $\mathit{Value} :: \ldots$

2.8 The Hypertexts - Bringing It All Together

Basically the developed datamodel says that a hypertext is a collection of nodes and one or more networks connecting the nodes and a structure describing the organization of the parts that forms the hypertext.

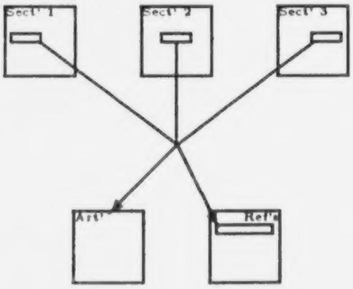


Figure 6: Example of N-ary links

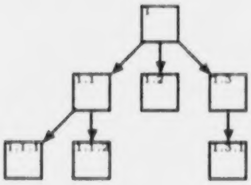


Figure 7: Example of a hierarchy

The networks represent the referential links, that is the explicit links connecting two or more parts of the hypertext. The structures are organizing the nodes and the networks. One can say that there is a dualism between networks and structures in that structures represent a kind of organizational links between nodes in a hypertext.

In this way one can represent several hypertext applications in a collection of nodes, simply by letting the actual hypertext application apply a certain network and a certain structure to the nodes. Then actual buttons in a node are first resolved by the hypertext application when one or more networks are applied to it and the node will show different sets of buttons depending on the applied networks. Finally a hypertext is defined as:

42.0 $Hypertext :: Nodes \times Networks \times Structures$

This observation leads to the object-oriented approach to a model, defining the hyperbase in terms of abstract datatypes, as presented in the following section.

3 An Object-Oriented Model

Having seen the basic datamodel of hypertext it clearly seems to be an good idea to follow an object-oriented approach in the specification of the semantic functions. Nodes, networks, and structures should be defined as abstract datatypes. The domains of each of these datatypes has already been described in the previous section.

In the following a simple model of an object-oriented database is presented. Based upon this model the operations of the abstract datatypes, as introduced by the datamodel in the previous section, is formally specified.

3.1 An Informal Model of an Object-Oriented Database

The *class* of an object is the abstract data type of the objects. Thus an object may be thought of as an instance of a particular class. The class defines the *operations* that can be applied to the object by an application. A class defines the set of operations applicable to all instances of that class in terms of names of operations and types of formal arguments and results. An implementation of a class provides a set of operation procedures implementing the set of operations defined by the class. The implementation encapsulates the data representation and the algorithms that are used to perform the operations. The data represention of an object is a collection of data that makes up the state of the object. The state is managed by the implementation and is only accessible by means of the operation procedures [Crawley 1986].

Below the basic domain of an object-oriented database is modelled as a collection of instantiated objects each having an unique identity. An instantiated object has a state that can be changed through the set of class operations. The domain of the state and the set of class operations are defined by the type definition of the class.

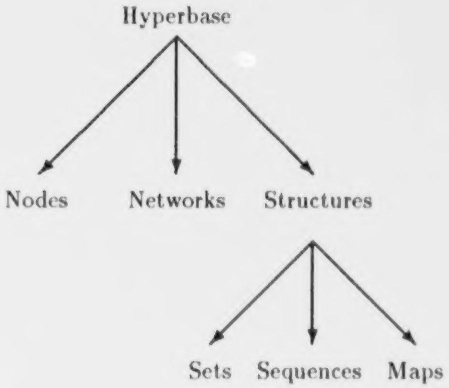


Figure 8: The Class Hierarchy

43.0 $Objectbase = Objid \xrightarrow{m} Object$
44.0 $Object :: State \times Opes$
45.0 $State = \ldots$
46.0 $Opes = Opeid \xrightarrow{m} Ope$
47.0 $Ope = Args \xrightarrow{\sim} State \xrightarrow{\sim} (State \times Res)$
48.0 $Args, Res = \ldots$

3.2 An Object-Oriented Hyperbase

Now the domain of hyperbases are applied to the concepts of object-oriented databases. The hyperbase covers basic operations on instances as the creation of new instances, basic object version management and object access control.

An object-oriented hyperbase is in this way defined as a collection of uniquely named instances of three object types. Each instance has a state which type depends on the type of the object. The three applicable state type are node, network and structure, as defined in the datamodel. A set of operations are defined for each type. Furthermore each instance has a set of predecessors and successors, identifying the neighbours of the instance in the version chain.

49.0 $HyperBase = Objid \xrightarrow{m} Object$
50.0 $Object :: State \times Operations \times Attributes \times Succ\text{-}\underline{set} \times Pred\text{-}\underline{set}$
51.0 $State = Node \mid Links \mid Structure$
52.0 $Objid = Nid \mid Nwid \mid Sid$
53.0 $Operations = Opeid \xrightarrow{m} Operation$
54.0 $Operation = Argument\text{-}\underline{set} \xrightarrow{\sim} State \xrightarrow{\sim} (State \times Result\text{-}\underline{set})$
55.0 $Opeid :: \mathrm{TOKEN}$

3.2.1 Fundamental Operations

The *CreateInstanceOf* operation can make instances of the subclasses, that is, it can make node, network and structural objects, returning the unique names of these objects. These instances can be destroyed by the *DestroyInstance* operation. The collection of identities of instances of a given class can be collected by the *SetOfInstances* operation.

56.0 $ObjectClass = \underline{\mathrm{NODES}} \mid \underline{\mathrm{NETWORKS}} \mid \underline{\mathrm{STRUCTURES}}$

57.0 type: *CreateInstanceOf : ObjectClass* $\overset{\sim}{\to}$ *Hyperbase* $\overset{\sim}{\to}$ *(Objid* × *Hyperbase)*
.1 type: *DestroyInstance : Objid* $\overset{\sim}{\to}$ *Hyperbase* $\overset{\sim}{\to}$ *Hyperbase*
.2 type: *SetOfInstances : ObjectClass* $\overset{\sim}{\to}$ *Hyperbase* $\overset{\sim}{\to}$ *Objid*-set

3.2.2 Basic Object Version Mangagement

This set of functions refer to the version management of the hyperbase. The *CreateSuccessorOfInstance* creates a copy of a specified object instance. The identity of the created object instance in added to the successor set of the specified instance, which identity on the other hand is added to the predecessor set of the new object instance. The predecessor and successor sets of an instance are found respectively by the *PredecessorOfInstance* and *SuccessorOfInstance* operations. The *MergeInstances* operation merge two objects into one object.

58.0 type: *CreateSuccessorOfInstance : Objid* $\overset{\sim}{\to}$ *Hyperbase* $\overset{\sim}{\to}$ *(HyperBase* × *Objid)*
.1 type: *PredecessorOfInstance : Objid* $\overset{\sim}{\to}$ *Hyperbase* $\overset{\sim}{\to}$ *Objid*-set
.2 type: *SuccessorOfInstance : Objid* $\overset{\sim}{\to}$ *Hyperbase* $\overset{\sim}{\to}$ *Objid*-set
.3 type: *MergeInstances : Objid* × *Objid* $\overset{\sim}{\to}$ *Hyperbase* $\overset{\sim}{\to}$ *(HyperBase* × *Objid)*

3.2.3 Object Access Control

The *Open* operation are concerned with checking the access conditions of the instance before allowing access to the set of operations. The *close* operation reset the access conditions after they have been altered by a previous *open*. One has access to the operations of the hyperbase objects through the *OperateOnInstance* function. The identity of the object instance and the name of the operation to be executed is passed to this function.

59.0 type: *Open : ...*
.1 type: *Close : ...*
.2 type: *OperateOnInstance : Objid*× *Opeid*× *Argument*-set$\overset{\sim}{\to}$*HyperBase*$\overset{\sim}{\to}$*(HyperBase*× *Result*-set*)*

3.2.4 Object Attribute Operations

AddAttribute adds an named attribute to the set of attributes of the slot. Attributes are removed by the *RemoveAttribute* operation. Values are assigned to attributes by the *AssignAttribute* operation. Finally a value of an attribute is read by using *ReadAttribute*.

60.0 type: *AddAttribute : (Objid* × *Name)* $\overset{\sim}{\to}$ *Node* $\overset{\sim}{\to}$ *Node*
.1 type: *RemoveAttribute : (Objid* × *Name)* $\overset{\sim}{\to}$ *Node* $\overset{\sim}{\to}$ *Node*
.2 type: *AssignAttribute : (Objid* × *Name* × *Value)* $\overset{\sim}{\to}$ *Node* $\overset{\sim}{\to}$ *Node*
.3 type: *ReadAttribute : (Objid* × *Name)* $\overset{\sim}{\to}$ *Node* $\overset{\sim}{\to}$ *Value*

3.3 The Three Object Classes of a Hyperbase

The three object classes or abstract datatypes of a hyperbase represent the nodes, the networks and the structures.

3.3.1 A Node Class

The objects of the node class are having zero or more slots. The operations are divided into three groups. The first set of operations is grouped around the schema of the node, and the second set is grouped around the end-point of links: handles and regions. The final group of operations is the node attributes operations.

Slot Operations. The *AddSlot* operation adds a new and empty slot to the node instance. The identity of the new slot is returned to the user. The *RemoveSlot* operation can remove a slot and its contents from the node. One can use the *ReturnSlots* operation to get set of names of the slots allocated in the schema of a node instance.

61.0 $\underline{\text{type}}: AddSlot : () \overset{\sim}{\rightarrow} Node \overset{\sim}{\rightarrow} (Node \times Slid)$
.1 $\underline{\text{type}}: RemoveSlot : Slid \overset{\sim}{\rightarrow} Node \overset{\sim}{\rightarrow} Node$
.2 $\underline{\text{type}}: ReturnSlots : () \overset{\sim}{\rightarrow} Node \overset{\sim}{\rightarrow} Slid\text{-}\underline{\text{set}}$

Slot Browsing Operations. The contents of a specified slot can be delivered as a string of characters by using *SlotView*. *SlotInsert* is an example of an editing operation. One can use this operation for insertion of a string into a position in the contents of a specified slot. *SlotDelete* can be used to remove a specified portion text of the contents of a slot.

62.0 $\underline{\text{type}}: SlotView : Slid \overset{\sim}{\rightarrow} Node \overset{\sim}{\rightarrow} \text{STRING}$
.1 $\underline{\text{type}}: SlotInsert : (Slid \times \text{STRING} \times Position) \overset{\sim}{\rightarrow} Node \overset{\sim}{\rightarrow} Node$
.2 $\underline{\text{type}}: SlotDelete : (Slid \times Position \times Length) \overset{\sim}{\rightarrow} Node \overset{\sim}{\rightarrow} (Node \times Hid\text{-}\underline{\text{set}})$

Handle Operations. A handle can be added to a specified region of the contents of a slot by the *AddHandle* operation. The handle is given a unique identity which is returned to the user. One can add several handles to the same region, and regions can be overlapping. A handle is removed by using *RemoveHandle*. The names of the handles located in a slot are returned by *ReturnPositionHandles* operation, and the names of the handles at a specified position in a slot is returned by the *ReturnPositionHandles* operation. The region specified by a handle is returned by the *GetHandle* operation.

63.0 $\underline{\text{type}}: AddHandle : (Slid \times Region) \overset{\sim}{\rightarrow} Node \overset{\sim}{\rightarrow} (Node \times Hid)$
.1 $\underline{\text{type}}: RemoveHandle : (Slid \times Hid) \overset{\sim}{\rightarrow} Node \overset{\sim}{\rightarrow} Node$
.2 $\underline{\text{type}}: ReturnSlotHandles : Slid \overset{\sim}{\rightarrow} Node \overset{\sim}{\rightarrow} Hid\text{-}\underline{\text{set}}$
.3 $\underline{\text{type}}: ReturnPositionHandles : (Slid \times Position) \overset{\sim}{\rightarrow} Node \overset{\sim}{\rightarrow} Hid\text{-}\underline{\text{set}}$
.4 $\underline{\text{type}}: GetHandle : (Slid \times Hid) \overset{\sim}{\rightarrow} Node \overset{\sim}{\rightarrow} Region$

The Slot Attribute Operations. *AddAttribute* adds an named attribute to the set of attributes of the slot. Attributes are removed by the *RemoveAttribute* operation. Values are assigned to attributes by the *AssignAttribute* operation. Finally a value of an attribute is read by using *ReadAttribute*.

64.0 $\underline{\text{type}}: AddAttribute : (Slid \times Name) \overset{\sim}{\rightarrow} Node \overset{\sim}{\rightarrow} Node$
.1 $\underline{\text{type}}: RemoveAttribute : (Slid \times Name) \overset{\sim}{\rightarrow} Node \overset{\sim}{\rightarrow} Node$
.2 $\underline{\text{type}}: AssignAttribute : (Slid \times Name \times Value) \overset{\sim}{\rightarrow} Node \overset{\sim}{\rightarrow} Node$
.3 $\underline{\text{type}}: ReadAttribute : (Slid \times Name) \overset{\sim}{\rightarrow} Node \overset{\sim}{\rightarrow} Value$

3.3.2 A Network Class

The operations of the network class consists of six network changing operations and three querying operations.

Network Changing Operations. The *AddLink* operation adds a new and empty link to the network. The operation gives the link a unique identity which is returned to the user. A link i removed by the *RemoveLink* operation. The anchors and destinations of the link in question, does not have to be empty. Anchors and destinations are added to a specified link by the two operations: *AddAnchor* and *AddDestination*. Removing anchors or destinations are done by the *RemoveAnchor* and *RemoveDestination* operations.

65.0 type: $AddLink : () \overset{\sim}{\to} Links \overset{\sim}{\to} (Links \times Lid)$
.1 type: $RemoveLink : Lid \overset{\sim}{\to} Links \overset{\sim}{\to} Links$
.2 type: $AddAnchor : (Lid \times Anchor) \overset{\sim}{\to} Links \overset{\sim}{\to} Links$
.3 type: $RemoveAnchor : (Lid \times Anchor) \overset{\sim}{\to} Links \overset{\sim}{\to} Links$
.4 type: $AddDestination : (Lid \times Destination) \overset{\sim}{\to} Links \overset{\sim}{\to} Links$
.5 type: $RemoveDestination : (Lid \times Destination) \overset{\sim}{\to} Links \overset{\sim}{\to} Links$

Network Querying Operations. The two querying operations *HavingAnchor* and *HavingDestination* are used to identify the links of a certain network instance, that have the specified anchors/destination in common. The *ReadLink* operation reads the anchor and destination set of the specified link.

66.0 type: $HavingAnchor : Anchor \overset{\sim}{\to} Links \overset{\sim}{\to} Lid\text{-}\underline{set}$
.1 type: $HavingDestination : Destination \overset{\sim}{\to} Links \overset{\sim}{\to} Lid\text{-}\underline{set}$
.2 type: $ReadLink : Lid \overset{\sim}{\to} Links \overset{\sim}{\to} (Anchor\text{-}\underline{set} \times Destination\text{-}\underline{set})$

The Link Attribute Operations. *AddAttribute* adds an named attribute to the set of attributes of the specified link. Attributes are removed by the *RemoveAttribute* operation. Values are assigned to attributes by the *AssignAttribute* operation. Finally a value of an attribute is read by using *ReadAttribute*.

67.0 type: $AddAttribute : (Lid \times Name) \overset{\sim}{\to} Links \overset{\sim}{\to} Links$
.1 type: $RemoveAttribute : (Lid \times Name) \overset{\sim}{\to} Links \overset{\sim}{\to} Links$
.2 type: $AssignAttribute : (Lid \times Name \times Value) \overset{\sim}{\to} Links \overset{\sim}{\to} Links$
.3 type: $ReadAttribute : (Lid \times Name) \overset{\sim}{\to} Links \overset{\sim}{\to} Value$

3.3.3 A Structural Class.

The operations of a structure are divided into four groups. The first is concerned the more general operations on the structure, i.e. adding and removing substructures etc. The final three groups are concerned with the specific operations of the three types of substructures: sets, sequences and maps.

Structure Operations A substructure can be added to the structure by using the *AddSubstructure* operation. A substructure is removed by *RemoveSubstructure*. The of identities of the substructures pointing the specified destination is returned by the *HavingDestination* operation. Finally, one can get the type of a substructure by using the *GetSubstructureType* operation.

68.0 $SubstructureType = \underline{\text{SET}} \mid \underline{\text{SEQUENCE}} \mid \underline{\text{MAP}}$

69.0 type: $AddSubstructure : SubstructureType \overset{\sim}{\to} Structure \overset{\sim}{\to} (Structure \times Subsid)$
.1 type: $RemoveSubstructure : Subsid \overset{\sim}{\to} Structure \overset{\sim}{\to} Structure$
.2 type: $HavingDestination : Destination \overset{\sim}{\to} Structure \overset{\sim}{\to} Subsid\text{-}\underline{set}$
.3 type: $GetSubstructureType : Subsid \overset{\sim}{\to} Structure \overset{\sim}{\to} SubstructureType$

Set Operations The *AddDestination* operation adds a destination to a set of destination. A destination element of a set i removed by *RemoveDestination*. The *HavingDestinationSet* operation can be used to find out whether a specified destination is in the set. The set of destinations is returned by the *GetDestinationSet* operation. One get the number of elements in the set by using the *GetCardinality* operation.

70.0 type: $AddDestination : (Subsid \times Destination) \overset{\sim}{\to} Structure \overset{\sim}{\to} Structure$
.1 type: $RemoveDestination : (Subsid \times Destination) \overset{\sim}{\to} Structure \overset{\sim}{\to} Structure$
.2 type: $HavingDestinationSet : (Subsid \times Destination) \overset{\sim}{\to} Structure \overset{\sim}{\to} \text{BOOL}$
.3 type: $GetDestinationSet : Subsid \overset{\sim}{\to} Structure \overset{\sim}{\to} Destination\text{-}\underline{set}$
.4 type: $GetCardinality : Subsid \overset{\sim}{\to} Structure \overset{\sim}{\to} N_0$

Sequence Operations. One can insert a destination at the specified position in the list by using the *InsertDestination* operation. Destinations positioned at a position greater or equal to the insertion point, are shifted one place. By the *RemoveDestination* operation one can remove the destination at the specified position. The operation works in the opposite way of the inserting operation. The operation returns all the positions of the specified destination in the sequence. The destination at the specified position is returned by *GetDestination*. *GetLength* returns the length, i.e. the number of destinations in the list.

71.0 type: $\mathit{InsertDestination} : (\mathit{Subsid} \times \mathit{Destination} \times \mathrm{N}_0) \xrightarrow{\sim} \mathit{Structure} \xrightarrow{\sim} \mathit{Structure}$
.1 type: $\mathit{RemoveDestination} : (\mathit{Subsid} \times \mathrm{N}_0) \xrightarrow{\sim} \mathit{Structure} \xrightarrow{\sim} \mathit{Structure}$
.2 type: $\mathit{HavingDestination} : (\mathit{Subsid} \times \mathit{Destination}) \xrightarrow{\sim} \mathit{Structure} \xrightarrow{\sim} \mathrm{N}_0\text{-}\underline{\mathrm{set}}$
.3 type: $\mathit{GetDestination} : (\mathit{Subsid} \times \mathrm{N}_0) \xrightarrow{\sim} \mathit{Structure} \xrightarrow{\sim} \mathit{Destination}$
.4 type: $\mathit{GetLength} : \mathit{Subsid} \xrightarrow{\sim} \mathit{Structure} \xrightarrow{\sim} \mathrm{N}_0$

Map Operations. A new named destination is added by the *AddDestination* operation and removed by the *RemoveDestination*. All the names of a specified destination can be found by *HavingDestination*. One can get the destination identified by a given name by using the *GetDestination* operation. The set of names bound to destinations is returned by *GetDomain*.

72.0 type: $\mathit{AddDestination} : (\mathit{Subsid} \times \mathit{Name} \times \mathit{Destination}) \xrightarrow{\sim} \mathit{Structure} \xrightarrow{\sim} \mathit{Structure}$
.1 type: $\mathit{RemoveDestination} : (\mathit{Subsid} \times \mathit{Name}) \xrightarrow{\sim} \mathit{Structure} \xrightarrow{\sim} \mathit{Structure}$
.2 type: $\mathit{HavingDestination} : (\mathit{Subsid} \times \mathit{Destination}) \xrightarrow{\sim} \mathit{Structure} \xrightarrow{\sim} \mathit{Name}\text{-}\underline{\mathrm{set}}$
.3 type: $\mathit{GetDestination} : (\mathit{Subsid} \times \mathit{Name}) \xrightarrow{\sim} \mathit{Structure} \xrightarrow{\sim} \mathit{Destination}$
.4 type: $\mathit{GetDomain} : \mathit{Subsid} \xrightarrow{\sim} \mathit{Structure} \xrightarrow{\sim} \mathit{Name}\text{-}\underline{\mathrm{set}}$

The Structure Attribute Operations. *AddAttribute* adds an named attribute to the set of attributes of the structure. Attributes are removed by the *RemoveAttribute* operation. Values are assigned to attributes by the *AssignAttribute* operation. Finally a value of an attribute is read by using *ReadAttribute*.

73.0 type: $\mathit{AddAttribute} : (\mathit{Subsid} \times \mathit{Name}) \xrightarrow{\sim} \mathit{Structure} \xrightarrow{\sim} \mathit{Structure}$
.1 type: $\mathit{RemoveAttribute} : (\mathit{Subsid} \times \mathit{Name}) \xrightarrow{\sim} \mathit{Structure} \xrightarrow{\sim} \mathit{Structure}$
.2 type: $\mathit{AssignAttribute} : (\mathit{Subsid} \times \mathit{Name} \times \mathit{Value}) \xrightarrow{\sim} \mathit{Structure} \xrightarrow{\sim} \mathit{Structure}$
.3 type: $\mathit{ReadAttribute} : (\mathit{Subsid} \times \mathit{Name}) \xrightarrow{\sim} \mathit{Structure} \xrightarrow{\sim} \mathit{Value}$

4 Conclusion

One of the major decisions in the development of this model has been to separate the presentation and the browsing semantics from the model, and move them to the applications design. The applications should only operate on the hyperbase through the specified operations and the dataobjects should not be aware of the applications and their semantics. By adding the aspects of persistence to this object-oriented model we have a model of an object-oriented database. In this way issues on distribution, basic version management and access control could be solved in the domain of object management systems. It is our intention to combine this model with the european standard on portable common tool environments (PCTE) [Thomas 1989]. PCTE is a standard for object-oriented bases for software engineering environments.

We are currently making a prototype of a hyperbase server based on the set of specifications presented here. This prototype is developed in the object-oriented programming language C++. Different hypertext applications are being developed for this server to show feasability of the model.

With respect to the work on hypertext standardization, this model should be related to existing approaches to hypertext, to seek for commonality between different approaches and to make progress towards a complete model. It is our opinion that a hypertext standard should be defined in terms of abstract datatypes, to retain a maximum of representational abstraction from the viewpoint of the hypertext applications. An open point in the model is the interchange mechanisms between different hyperbases. The model has to be extended with some kind of protocol for the transfer of hypertexts from one base to another.

References

[Bjørner & Jones 1982] Bjørner, D., Jones, C.B. *Formal Specification & Software Development.* Prentice-Hall International 1982.

[Consens & Mendelzon 1989] Consens, M.P., Mendelzon, A.D. Expressing Structural Hypertext Queries in GraphLog. In *Hypertext'89 Proceedings.* Pittsburgh, Pennsylvania, USA. November 1989.

[Crawley 1986] Crawley, S. An Object-Based File System for Large Scale Applications. In *Software Engineering Environments*, ed. Ian Sommerville. Peter Peregrinus Ltd., 1986.

[Delisle & Schwartz 1987] Delisle, N.M., Schwartz, M.D. Contexts - A Partitioning Concept for Hypertext. *ACM TOOIS 5*, 2, pp168-186, 1987.

[Garg 1988] Garg, P.K. Abstraction Mechanisms in Hypertext. *Communications of the ACM, 31*, 7, pp862-870, 1988.

[Güting et al.] Güting, R.H., Zicari, R., Choy, D.M. An Algebra for Structured Office Documents. *ACM TOOIS,* 7, 4, pp123-157, 1989.

[Halasz & Conklin 1989] Halasz, F., Conklin, J. *Issues in the Design and Application of Hypermedia Systems.* Tutorial at SIGCHI 89, Austin, Texas, 1989.

[Hypertext 1989] *Hypertext'89 Proceeding.* Pittsburgh, Pennsylvania, USA. November 1989.

[Jones 1986] Jones, C.B. *Systematic Software Development Using VDM.* Prentice-Hall International 1986

[Steele 1984] Steele Jr., G.L. COMMON LISP *The Language.* Digital Press, 1984.

[Stotts & Furuta 1989] Stotts, P.D., Furuta, R. Petri Net Based Hypertext: Document Structure with Browsing Semantics. *ACM TOOIS,* 7, 1, pp3-29, 1989.

[Thomas 1989] Thomas, I. PCTE Interfaces: Supporting Tools in Software Engineering Environments. *IEEE Software, 6,* 6, pp15-23, 1989.

A Detail Specifications

A.1 An Object-Oriented Hyperbase

```
74.0  type: CreateInstanceOf : ObjectClass ⥲ Hyperbase ⥲ (Objid × Hyperbase)
  .1  pre-CreateInstanceOf(class, ) ≜ class ∈ {NODES, NETWORKS, STRUCTURES}
  .2  post-CreateInstanceOf(class, hyperbase)(objid, hyperbase')) ≜
  .3      let objid ∈ Objid \ dom hyperbase in
  .4          cases class :
  .5              NODES          → hyperbase'= hyperbase ∪ [mk-Nid(objid) ↦
  .6                      mk-Object([], NodeOperations, [], { }, { }],
  .7              NETWORKS       → hyperbase'= hyperbase ∪ [mk-Nwid(objid) ↦
  .8                      mk-Object([], LinksOperations, [], { }, { }],
  .9              STRUCTURES     → hyperbase'= hyperbase ∪ [mk-Sid(objid) ↦
  .10                     mk-Object([], StructureOperations, [], { }, { }],
75.0  type: DestroyInstance : Objid ⥲ Hyperbase ⥲ Hyperbase
  .1  pre-DestroyInstance(objid, hyperbase) ≜ objid ∈ dom hyperbase
  .2  post-DestroyInstance(objid, hyperbase)(hyperbase')) ≜
  .3      hyperbase' = [id ↦ (let mk-Object(state, operations, ss, ps) = hyperbase(id) in
  .4          mk-Object(state, operations,
  .5          (objid ∈ ss→ (ss \ {objid}) ∪ s-Succ(hyperbase(objid)),
  .6          T          → ss),
  .7          (objid ∈ ps→ (ps \ {objid}) ∪ s-Pred(hyperbase(objid)),
  .8          T          → ps)))]
```

```
76.0  type: SetOfInstances : ObjectClass ⥲ Hyperbase ⥲ Objid-set
  .1  pre-SetOfInstances(class, ) ≜ class ∈ {NODES, NETWORKS, STRUCTURES}
  .2  post-SetOfInstances(class, hyperbase)(objids)) ≜
  .3      cases class :
  .4          NODES       → objids = {objid | (∀ objid ∈ dom hyperbase)(is-Node(objid))}
  .5          NETWORKS    → objids = {objid | (∀ objid ∈ dom hyperbase)(is-Links(objid))}
  .6          STRUCTURES  → objids = {objid | (∀ objid ∈ dom hyperbase)(is-Structures(objid))}
```

A.1.1 Basic Object Version Mangagement

```
77.0  type: CreateSuccessorOfInstance : Objid ⥲ Hyperbase ⥲ (HyperBase × Objid)
  .1  pre-CreateSuccessorOfInstance(objid, hyperbase) ≜ objid ∈ dom hyperbase
  .2  post-CreateSuccessorOfInstance(objid, hyperbase)(hyperbase', objid')) ≜
  .3      let objid'∈ Objid \ dom hyperbase in
  .4          let mk-Object(state, operations, attrs, ss, ps) = hyperbase(objid) in
  .5              hyperbase'=hyperbase+[objid↦mk-Object(state,operations,attrs,ss ∪ {objid'}, ps)]
  .6                                  ∪ [objid'↦ mk-Object(state, operations, attrs, { },{objid})]
```

```
78.0  type: PredecessorOfInstance : Objid ⥲ Hyperbase ⥲ Objid-set
  .1  pre-PredecessorOfInstance(objid, hyperbase) ≜ objid ∈ dom hyperbase
  .2  post-PredecessorOfInstance(objid, hyperbase)(objids) ≜ objids = s-Pred(hyperbase(obid))
```

```
79.0  type: SuccessorOfInstance : Objid ⥲ Hyperbase ⥲ Objid-set
  .1  pre-SuccessorOfInstance() ≜ objid ∈ dom hyperbase
  .2  post-SuccessorOfInstance(objid, hyperbase)(objids) ≜ objids = s-Succ(hyperbase(obid))
```

```
80.0  type: MergeInstances : ...
```

A.1.2 Object Access Control

```
81.0  type: Open : ...
```

```
82.0  type: Close : ...
```

```
83.0  type: OperateOnInstance : Objid× Opeid× Argument-set⥲HyperBase⥲(HyperBase× Result-set)
  .1  pre-OperateOnInstance(objid, opeid, , hyperbase) ≜
  .2      objid ∈ dom hyperbase ∧ opeid ∈ dom s-Operations(hyperbase(objid))
  .3  post-OperateOnInstance(objid, opeid, as, hyperbase)(hyperbase', rs') ≜
  .4      let mk-Object(state, operations, attrs, ss, ps) = hyperbase(objid) in
  .5          let (state', rs') = operations(opeid)(as, state) in
  .6              (state'≠ nil →
  .7                  hyperbase'= hyperbase + [objid ↦ mk-Object(state', operations, attrs, ss, ps)],
  .8              state'= nil → hyperbase'= hyperbase)
```

Object Attribute Operations.

84.0 $\underline{\text{type}}\colon \mathit{AddAttribute} : \ldots$

85.0 $\underline{\text{type}}\colon \mathit{RemoveAttribute} : \ldots$

86.0 $\underline{\text{type}}\colon \mathit{AssignAttribute} : \ldots$

87.0 $\underline{\text{type}}\colon \mathit{ReadAttribute} : \ldots$

A.2 The Three Object Classes of a Hyperbase

A.2.1 A Node Class

Schema Operations.

88.0 $\underline{\text{type}}\colon \mathit{AddSlot} : () \overset{\sim}{\rightarrow} \mathit{Node} \overset{\sim}{\rightarrow} (\mathit{Node} \times \mathit{Slid})$
.1 $\underline{\text{pre-}}\mathit{AddSlot}() \triangleq \mathsf{T}$
.2 $\underline{\text{post-}}\mathit{AddSlot}(\mathit{node})(\mathit{node}', \mathit{slid}) \triangleq$
.3 $\quad \underline{\text{let}}\ \mathit{slid} \in \mathit{Slid} \setminus \underline{\text{dom}}\ \mathit{node}\ \underline{\text{in}}\ \mathit{node}' = \mathit{node} \cup [\mathit{slid} \mapsto \underline{\text{mk-}}\mathit{Slot}(< >, [\,], [\,])]$

89.0 $\underline{\text{type}}\colon \mathit{RemoveSlot} : \mathit{Slid} \overset{\sim}{\rightarrow} \mathit{Node} \overset{\sim}{\rightarrow} \mathit{Node}$
.1 $\underline{\text{pre-}}\mathit{RemoveSlot}(\mathit{slid}, \mathit{node}) \triangleq \mathit{slid} \in \underline{\text{dom}}\ \mathit{node}$
.2 $\underline{\text{post-}}\mathit{RemoveSlot}(\mathit{slid}, \mathit{node})(\mathit{node}') \triangleq \mathit{node}' = \mathit{node} \setminus \{\mathit{slid}\}$

90.0 $\underline{\text{type}}\colon \mathit{ReturnSlots} : () \overset{\sim}{\rightarrow} \mathit{Node} \overset{\sim}{\rightarrow} \mathit{Slid}\underline{\text{-set}}$
.1 $\underline{\text{pre-}}\mathit{ReturnSlots}() \triangleq \mathsf{T}$
.2 $\underline{\text{post-}}\mathit{ReturnSlots}(\mathit{node})(\mathit{slids}) \triangleq \mathit{slids} = \underline{\text{dom}}\ \mathit{node}$

Slot Browsing Operations.

91.0 $\underline{\text{type}}\colon \mathit{SlotView} : \mathit{Slid} \overset{\sim}{\rightarrow} \mathit{Node} \overset{\sim}{\rightarrow} \mathit{String}$
.1 $\underline{\text{pre-}}\mathit{SlotView}(\mathit{slid}, \mathit{node}) \triangleq \mathit{slid} \in \underline{\text{dom}}\ \mathit{node}$
.2 $\underline{\text{post-}}\mathit{SlotView}(\mathit{slid}, \mathit{node})(\mathit{text}) \triangleq$
.3 $\quad \underline{\text{let}}\ \underline{\text{mk-}}\mathit{Slot}(\mathit{string}, ,) = \mathit{node}(\mathit{slid})\ \underline{\text{in}}\ \mathit{text} = \mathit{string}$

92.0 $\underline{\text{type}}\colon \mathit{SlotInsert} : (\mathit{Slid} \times \mathit{String} \times \mathit{Position}) \overset{\sim}{\rightarrow} \mathit{Node} \overset{\sim}{\rightarrow} \mathit{Node}$
.1 $\underline{\text{pre-}}\mathit{SlotInsert}(\mathit{slid}, , \mathit{position}, \mathit{node}) \triangleq$
.2 $\quad \mathit{slid} \in \underline{\text{dom}}\ \mathit{node} \wedge (\underline{\text{let}}\ \underline{\text{mk-}}\mathit{Slot}(\mathit{str}, ,) = \mathit{node}(\mathit{slid})\ \underline{\text{in}}\ 0 \leq \mathit{position} \leq \underline{\text{len}}\ \mathit{str})$
.3 $\underline{\text{post-}}\mathit{SlotInsert}(\mathit{slid}, s, \mathit{position}, \mathit{node})\ (\mathit{node}') \triangleq$
.4 $\quad (\underline{\text{let}}\ \underline{\text{mk-}}\mathit{Slot}(\mathit{text}, \mathit{handles}, \mathit{attrs}) = \mathit{node}(\mathit{slid}),$
.5 $\quad\quad \underline{\text{mk-}}\mathit{Slot}(\mathit{text}', \mathit{handles}', \mathit{attrs}') = \mathit{node}'(\mathit{slid})\ \underline{\text{in}}$
.6 $\quad\quad \mathit{text}' = < \mathit{text}[i] \mid 0 \leq i < \mathit{position} > \frown s \frown < \mathit{text}[i] \mid \mathit{position} \leq i < \underline{\text{len}}\ \mathit{text} > \wedge$
.7 $\quad\quad (\forall\, \mathit{hid} \in \underline{\text{dom}}\ \mathit{handles})\ (\underline{\text{let}}\ (p, l) = \mathit{handles}(\mathit{hid}), (p', l') = \mathit{handles}'(\mathit{hid})\ \underline{\text{in}}$
.8 $\quad\quad\quad\quad\quad\quad p + l < \mathit{position} \quad\quad \rightarrow p' = p \wedge l' = l,$
.9 $\quad\quad\quad\quad\quad\quad p < \mathit{position} \geq p + l \rightarrow p' = p \wedge l' = l + \mathit{length},$
.10 $\quad\quad\quad\quad\quad\quad \mathit{position} > p \quad\quad\quad \rightarrow p' = p + \mathit{length} \wedge l' = l$

93.0 $\underline{\text{type}}\text{: } SlotDelete : (Slid \times Position \times Length) \xrightarrow{\sim} Node \xrightarrow{\sim} (Node \times Hid\text{-}\underline{\text{set}})$
.1 $\underline{\text{pre-}}SlotDelete(slid, position, , node) \triangleq$
.2 $\quad slid \in \underline{\text{dom}}\ node \wedge$
.3 $\quad (\underline{\text{let}}\ \underline{\text{mk-}}Slot(str, ,) = node(slid)\ \underline{\text{in}}$
.4 $\qquad 0 \leq position \leq \underline{\text{len}}\ str \wedge position + length < \underline{\text{len}}\ str)$
.5 $\underline{\text{post-}}SlotDelete(slid, position, length, node)\ (node', hids) \triangleq$
.6 $\quad (\underline{\text{let}}\ \underline{\text{mk-}}Slot(text, handles, attrs) = node(slid),$
.7 $\qquad \underline{\text{mk-}}Slot(text', handles', attrs) = node'(slid)\ \underline{\text{in}}$
.8 $\qquad text' = \langle text[i] \mid 0 \leq i < positio \rangle\ ^\frown\ \langle text[i] \mid position + length \leq i < \underline{\text{len}}\ text \rangle \wedge$
.9 $\qquad hids = \{hid \mid (\forall\ hid \in \underline{\text{dom}}\ handles)\ (\underline{\text{let}}\ (p, l) = handles(hid)\ \underline{\text{in}}$
.10 $\qquad\qquad\qquad position \leq p \wedge position + length \geq p + l)\}\ )) \wedge$
.11 $\qquad \underline{\text{dom}}\ handles' = \underline{\text{dom}}\ handles \setminus hids \wedge$
.12 $\qquad position < p \wedge position + length < p \quad \rightarrow p' = p - position \wedge l' = l,$
.13 $\qquad position \leq p \wedge position + length < p+l \rightarrow p' = p - position \wedge l' = l - (position + length - p),$
.14 $\qquad p \leq position \wedge position + length < p+l \rightarrow p' = p \wedge l' = l - length,$
.15 $\qquad p < position \wedge p+l \leq position + length \rightarrow p' = p \wedge l' = l - (p+l - position),$
.16 $\qquad p+l < position \qquad\qquad\qquad \rightarrow p' = p \wedge l' = l$

Handle Operations.

94.0 $\underline{\text{type}}\text{: } AddHandle : (Slid \times Region) \xrightarrow{\sim} Node \xrightarrow{\sim} (Node \times Hid)$
.1 $\underline{\text{pre-}}AddHandle(slid, \underline{\text{mk-}}Region(pos, length), node) \triangleq$
.2 $\quad slid \in \underline{\text{dom}}\ node \wedge (\underline{\text{let}}\ \underline{\text{mk-}}Slot(str, ,) = node(slid)\ \underline{\text{in}}\ pos+length < \underline{\text{len}}\ str\ )$
.3 $\underline{\text{post-}}AddHandle(slid, region, node)(node', hid) \triangleq$
.4 $\quad \underline{\text{let}}\ \underline{\text{mk-}}Slot(text, handles, attrs) = node(slid),\ hid \in Hid \setminus \underline{\text{dom}}\ handles\ \underline{\text{in}}$
.5 $\qquad node' = node + [slid \mapsto \underline{\text{mk-}}Slot(text, handles \cup [hid \mapsto region], attrs)]$

95.0 $\underline{\text{type}}\text{: } RemoveHandle : (Slid \times Hid) \xrightarrow{\sim} Node \xrightarrow{\sim} Node$
.1 $\underline{\text{pre-}}RemoveHandle(slid, hid, node) \triangleq$
.2 $\quad slid \in \underline{\text{dom}}\ node \wedge (\underline{\text{let}}\ \underline{\text{mk-}}Slot(, handles,) = node(slid)\ \underline{\text{in}}\ hid \in \underline{\text{dom}}\ handles\ )$
.3 $\underline{\text{post-}}RemoveHandle(slid, hid, node)(node') \triangleq$
.4 $\quad \underline{\text{let}}\ \underline{\text{mk-}}Slot(text, handles, attrs) = node(slid)\ \underline{\text{in}}$
.5 $\qquad node' = node + [slid \mapsto \underline{\text{mk-}}Slot(text, handles \setminus \{hid'\}, attrs)]$

96.0 $\underline{\text{type}}\text{: } ReturnSlotHandles : Slid \xrightarrow{\sim} Node \xrightarrow{\sim} Hid\text{-}\underline{\text{set}}$
.1 $\underline{\text{pre-}}ReturnSlotHandles(slid, node) \triangleq slid \in \underline{\text{dom}}\ node$
.2 $\underline{\text{post-}}ReturnSlotHandles(slid, node)(hids) \triangleq hids = \underline{\text{dom}}\ \underline{\text{s-}}Handles(node(slid)))$

97.0 $\underline{\text{type}}\text{: } ReturnPositionHandles : (Slid \times Position) \xrightarrow{\sim} Node \xrightarrow{\sim} Hid\text{-}\underline{\text{set}}$
.1 $\underline{\text{pre-}}ReturnPositionHandles(slid, position, node) \triangleq$
.2 $\quad slid \in \underline{\text{dom}}\ node \wedge (\underline{\text{let}}\ \underline{\text{mk-}}Slot(str, ,) = node(slid)\ \underline{\text{in}}\ position < \underline{\text{len}}\ str\ )$
.3 $\underline{\text{post-}}ReturnPositionHandles(, position)(hids) \triangleq$
.4 $\quad \underline{\text{let}}\ \underline{\text{mk-}}Slot(, handles,) = node(slid)\ \underline{\text{in}}$
.5 $\qquad hids = \{hid \in \underline{\text{dom}}\ handles \mid (\underline{\text{let}}\ \underline{\text{mk-}}Region(p,l) = handles(hid)\ \underline{\text{in}}$
.6 $\qquad\qquad\qquad p \leq position \leq p+l)\}$

98.0 $\underline{\text{type}}\text{: } GetHandle : (Slid \times Hid) \xrightarrow{\sim} Node \xrightarrow{\sim} Region$
.1 $\underline{\text{pre-}}GetHandle(slid, hid, node) \triangleq slid \in \underline{\text{dom}}\ node \wedge hid \in \underline{\text{dom}}\ \underline{\text{s-}}Handles(ndoe(slid))$
.2 $\underline{\text{post-}}GetHandle(slid, hid, node)(region) \triangleq$
.3 $\quad \underline{\text{let}}\ \underline{\text{mk-}}Slot(, handles,) = node(slid)\ \underline{\text{in}}\ region = handles(hid)$

The Slot Attribute Operations.

99.0 $\underline{\text{type}}\text{: } AddAttribute : \ldots$

```
100.0   type: RemoveAttribute : ...

101.0   type: AssignAttribute : ...

102.0   type: ReadAttribute : ...
```

A.2.2 A Network Class

Network Changing Operations.

```
103.0   type: AddLink : () ⥲ Links ⥲ ( Links × Lid)
   .1   pre-AddLink() ≜ T
   .2   post-AddLink(links)(links', lid') ≜
   .3       let lid'∈ Lid \ dom links in links'= links ∪ [lid'↦ mk-Link(mk-Connections({ },{ }),[])]

104.0   type: RemoveLink : Lid ⥲ Links ⥲ Links
   .1   pre-RemoveLink(lid, links) ≜ lid ∈ dom links
   .2   post-RemoveLink(lid, links)(links') ≜ links'= links \ {lid}

105.0   type: AddAnchor : (Lid × Anchor) ⥲ Links ⥲ Links
   .1   pre-AddAnchor(lid, , links) ≜ lid ∈ dom links
   .2   post-AddAnchor(lid, anchor, links)(links') ≜
   .3       let mk-Link(mk-Connections(as, ds), attrs) = links(lid) in
   .4           links'= links + [lid ↦ mk-Link(mk-Connections(as ∪ {anchor}, ds), attrs)]

106.0   type: RemoveAnchor : (Lid × Anchor) ⥲ Links ⥲ Links
   .1   pre-RemoveAnchor(lid, anchor, links) ≜
   .2       lid ∈ dom links ∧ (let mk-Connections(as, ) = links(lid) in anchor ∈ as )
   .3   post-RemoveAnchor(lid, anchor, links)(links') ≜
   .4       let mk-Link(mk-Connections(as, ds), attrs) = links(lid) in
   .5           links'= links + [lid ↦ mk-Link(mk-Connections(as \ {anchor}, ds), attrs)]

107.0   type: AddDestination : (Lid × Destination) ⥲ Links ⥲ Links
   .1   pre-AddDestination(lid, destination, links) ≜ lid ∈ dom links
   .2   post-AddDestination(lid, destination, links)(links') ≜
   .3       let mk-Link(mk-Connections(as, ds), attrs) = links(lid) in
   .4           links'= links + [lid ↦ mk-Link(mk-Connections(as, ds ∪ {destination}), attrs)]

108.0   type: RemoveDestination : (Lid × Destination) ⥲ Links ⥲ Links
   .1   pre-RemoveDestination(lid, destination, links)(links') ≜
   .2       lid ∈ dom links ∧ (let mk-Connections( , ds ) = links(lid) in destination ∈ ds )
   .3   post-RemoveDestination(lid, destination, links)(links') ≜
   .4       let mk-Link(mk-Connections(as, ds), attrs) = links(lid) in
   .5           links'= links + [lid ↦ mk-Link(mk-Connections(as, ds \ {destination}), attrs)]
```

Network Querying Operations.

```
109.0   type: HavingAnchor : Anchor ⥲ Links ⥲ Lid-set
   .1   pre-HavingAnchor() ≜ T
   .2   post-HavingAnchor(anchor, links)(lids) ≜
   .3       lids = {lid ∈ dom links | let mk-Link(mk-Connections(as, ), ) = links(lid) in anchor ∈ as}
```

```
110.0  type: HavingDestination : Destination ⥲ Links ⥲ Lid-set
   .1  pre-HavingDestination() ≜ T
   .2  post-HavingDestination(destination, links)(lids) ≜
   .3      lids = {lid∈dom links|let mk-Link(mk-Connections(,ds), ) = links(lid) in destination ∈ ds}

111.0  type: ReadLink : Lid ⥲ Links ⥲ (Anchor-set × Destination-set)
   .1  pre-ReadLink(lid, links) ≜ lid ∈ dom links
   .2  post-ReadLink(lid, links)(as, ds) ≜ mk-Link(mk-Connections(as, ds), ) = links(lid)
```

The Link Attribute Operations.

```
112.0  type: AddAttribute : ...

113.0  type: RemoveAttribute : ...

114.0  type: AssignAttribute : ...

115.0  type: ReadAttribute : ...
```

A.2.3 A Structural Class.

Structure Operations

```
116.0  type: AddSubstructure : SubstructureType ⥲ Structure ⥲ (Structure × Subsid)
   .1  pre-AddSubstructure() ≜ T
   .2  post-AddSubstructure(type ,structure)(structure',subsid) ≜
   .3      let subsid ∈ Subsid \ dom structure in
   .4          structure'= structure ∪ [subsid ↦
   .5              mk-Substructure( (cases type :
   .6                  SET      → mk-Set({ }),
   .7                  SEQUENCE → mk-Seq(< >),
   .8                  MAP      → mk-Map([]) ), [])]

117.0  type: RemoveSubstructure : Subsid ⥲ Structure ⥲ Structure
   .1  pre-RemoveSubstructure(subsid, structure) ≜ subsid ∈ dom structure
   .2  post-RemoveSubstructure(subsid, structure)(structure') ≜ structure'= structure \ {subsid}

118.0  type: HavingDestination : Destination ⥲ Structure ⥲ Subsid-set
   .1  pre-HavingDestination() ≜ T
   .2  post-HavingDestination(destination, structures)(subsids) ≜
   .3      subsids = {subsid | (∀ subsid ∈ dom structure)
   .4      let mk-Substucture(substruc, ) = structure(subsid) in
   .5              cases substruc :
   .6                  SET      → destination ∈ s,
   .7                  SEQUENCE → destination ∈ elems s,
   .8                  MAP      → destination ∈ rng s)}
```

```
119.0  type: GetSubstructureType : Subsid ⥲ Structure ⥲ SubstructureType
   .1  pre-GetSubstructureType(subsid, structure) ≜ subsid ∈ dom structure
   .2  post-GetSubstructureType(subsid,structure)(type) ≜
   .3      let mk-Substucture(substruc, ) = structure(subsid) in
   .4          type = (cases substruc :
   .5              mk-Set()  → SET,
   .6              mk-Seq()  → SEQUENCE,
   .7              mk-Map() → MAP)
```

Set Operations

```
120.0  type: AddDestination : (Subsid × Destination) ⥲ Structure ⥲ Structure
   .1  pre-AddDestination(subsid, , structure) ≜
   .2      subsid ∈ dom structure ∧
   .3      let mk-Substructure(substruc, ) = structures(subsid) in is-Set(substruc)
   .4  post-AddDestination(subsid, destination, structure)(structure') ≜
   .5      let mk-Substructure(substruc, attrs) = structure(subsid) in
   .6          structure'= structure + [subsid ↦mk-Substructure(substruc∪{destination},attrs)]

121.0  type: RemoveDestination : (Subsid × Destination) ⥲ Structure ⥲ Structure
   .1  pre-RemoveDestination(subsid, destination, structure) ≜
   .2      subsid ∈ dom structure ∧
   .3      let mk-Substructure(substruc, ) = structures(subsid) in
   .4          is-Set(substruc) ∧ destination ∈ substruc
   .5  post-RemoveDestination(subsid,destination,structure)(structure')≜
   .6      let mk-Substructure(substruc, attrs) = structure(subsid) in
   .7          structure'= structure + [subsid ↦mk-Substructure(substruc \ {destination}, attrs)]

122.0  type: HavingDestinationSet : (Subsid × Destination) ⥲ Structure ⥲ BOOL
   .1  pre-HavingDestinationSet(subsid, , structure) ≜
   .2      subsid ∈ dom structure ∧
   .3      let mk-Substructure(substruc, ) = structures(subsid) in is-Set(substruc)
   .4  post-HavingDestinationSet(subsid, destination, structure)(b) ≜
   .5      let mk-Substructure(substruc, ) = structure(subsid) in b ⇔ destination ∈ substruc

123.0  type: GetDestinationSet : (Subsid) ⥲ Structure ⥲ Destination-set
   .1  pre-GetDestinationSet(subsid, structure) ≜
   .2      subsid ∈ dom structure ∧
   .3      let mk-Substructure(substruc, ) = structures(subsid) in is-Set(substruc)
   .4  post-GetDestinationSet(subsid, structure)(ds) ≜
   .5      let mk-Substructure(substruc, ) = structure(subsid) in ds = substruc

124.0  type: GetCardinality : Subsid ⥲ Structure ⥲ N0
   .1  pre-GetCardinality(subsid, structure) ≜
   .2      subsid ∈ dom substructure ∧
   .3      let substruc = s-Substruc(substructures(subsid)) in is-Set(substruc)
   .4  post-GetCardinality(subsid, structures)(cd) ≜
   .5      let mk-Substructure(substruc, ) = structure(subsid) in cd = card substruc
```

Sequence Operations.

```
125.0   type: InsertDestination : (Subsid × Destination × N₀) ⥲ Structure ⥲ Structure
   .1   pre-InsertDestination(subsid,  , index, structure) ≜
   .2       subsid ∈ dom substructure ∧
   .3       let mk-Substructure(substruc, ) = structure(subsid) in
   .4           is-Seq(substruc) ∧ 0 ≤ index ≤ lensubstruc
   .5   post-InsertDestination(subsid, destination, index, structures) (structure')≜
   .6       let mk-Substructure(substruc, attrs) = structure(subsid) in
   .7           structure'=structure+[subsid ↦mk-Substructure(<substruc[i]|0≤i<index> ⌢
   .8               <destination > ⌢ <substruc[i] | index ≤ i < len substruc>, attrs)]

126.0   type: RemoveDestination : (Subsid × N₀) ⥲ Structure ⥲ Structure
   .1   pre-RemoveDestination(subsid, index, structure) ≜
   .2       subsid ∈ dom substructure ∧
   .3       let mk-Substructure(substruc, ) = structure(subsid) in
   .4           is-Seq(substruc) ∧ 0 ≤ index ≤ lensubstruc
   .5   post-RemoveDestination(subsid, index, structure)(structure') ≜
   .6           let mk-Substructure(substruc, attrs) = structure(subsid) in
   .7               structure'=structure+[subsid ↦
   .8                       mk-Substructure(<substruc[i] | 0 ≤ i < index> ⌢
   .9                           <substruc[i] | index < i < len substruc>, attrs)]

127.0   type: HavingDestination : (Subsid × Destination) ⥲ Structure ⥲ N₀-set
   .1   pre-HavingDestination(subsid, destination, structure) ≜
   .2       subsid ∈ dom substructure ∧
   .3       let mk-Substructure(substruc, ) = structure(subsid) inis-Seq(substruc)
   .4   post-HavingDestination(subsid, destination, structure)(indices) ≜
   .5       let mk-Substructure(substruc, ) = structure(subsid) in
   .6           indices = {i | (i ∈ ind substruc)(substruc[i]= destination)}

128.0   type: GetDestination : (Subsid × N₀) ⥲ Structure ⥲ Destination
   .1   pre-GetDestination(subsid, index, structure) ≜
   .2       subsid ∈ dom substructure ∧
   .3       let mk-Substructure(substruc, ) = structure(subsid) in
   .4           is-Seq(substruc) ∧ 0 ≤ index ≤ lensubstruc
   .5   post-GetDestination(subsid, index, structure)(destination) ≜
   .6       let mk-Substructure(substruc, ) = structure(subsid) in destination = substruc[index]

129.0   type: GetLength : (Subsid) ⥲ Structure ⥲ N₀
   .1   pre-GetLength(subsid, structure) ≜
   .2       subsid ∈ dom substructure ∧
   .3       let mk-Substructure(substruc, ) = structure(subsid) inis-Seq(substruc)
   .4   post-GetLength(subsid, structure)(length) ≜
   .5       let mk-Substructure(substruc, ) = structures(subsid) in length = len substruc
```

Map Operations.

```
130.0  type: AddDestination : (Subsid × Name × Destination) ⥲ Structure ⥲ Structure
   .1  pre-AddDestination(subsid, name, , structure) ≜
   .2      subsid ∈ dom structure ∧
   .3      let mk-Substructure(substruc, ) = structure(subsid) in
   .4          is-Map(substruc) ∧ name ∉ dom substruc
   .5  post-AddDestination(subsid,name,destination,structure)(substructure')≜
   .6      let mk-Substructure(substruc, attrs) = structure(subsid) in
   .7          structure'=structure+[subsid↦mk-Substructure(substruc∪[name↦destination],attrs)]

131.0  type: RemoveDestination : (Subsid × Name) ⥲ Structure ⥲ Structure
   .1  pre-RemoveDestination(subsid, name, structure) ≜
   .2      subsid ∈ dom structure ∧
   .3      let mk-Substructure(substruc, ) = structure(subsid) in
   .4          is-Map(substruc) ∧ name ∈ dom substruc
   .5  post-RemoveDestination(subsid,name,structures)(structure')≜
   .6      let mk-Substructure(substruc, attrs) = structure(subsid) in
   .7          structure'=structure+[subsid ↦mk-Substructure(substruc \ {name}, attrs)]

132.0  type: HavingDestination : (Subsid × Destination) ⥲ Structure ⥲ Name-set
   .1  pre-HavingDestination(subsid, , structure) ≜
   .2      subsid ∈ dom structure ∧
   .3      let mk-Substructure(substruc, ) = structure(subsid) in is-Map(substruc)
   .4  post-HavingDestination(subsid, destination, structure) (names) ≜
   .5      let mk-Substructure(substruc, attrs) = structure(subsid) in
   .6          names = {name | (name ∈ dom substruc) (substruc(name) = destination)}

133.0  type: GetDestination : (Subsid × Name) ⥲ Structure ⥲ Destination
   .1  pre-GetDestination(subsid, name, structure) ≜
   .2      subsid ∈ dom structure ∧
   .3      let mk-Substructure(substruc, ) = structure(subsid) in is-Map(substruc)
   .4  post-GetDestination(subsid, name, structures) (destination) ≜
   .5      let mk-Substructure(substruc, attrs) = structure(subsid) in destination = substruc(name)

134.0  type: GetDomain : Subsid ⥲ Structure ⥲ Name-set
   .1  pre-GetDomain(subsid, structure) ≜
   .2      subsid ∈ dom structure ∧
   .3      let mk-Substructure(substruc, ) = structure(subsid) in is-Map(substruc)
   .4  post-GetDomain(subsid, structure) (ns) ≜
   .5      let mk-Substructure(substruc, attrs) = structure(subsid) in ns = dom substruc
```

The Structure Attribute Operations.

```
135.0  type: AddAttribute : ...

136.0  type: RemoveAttribute : ...

137.0  type: AssignAttribute : ...

138.0  type: ReadAttribute : ...
```

A Multi-Tiered Approach to Hypertext Integration: Negotiating Standards for a Heterogeneous Application Environment.

Catherine C. Marshall

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304

Submitted to the NIST Hypertext Standardization Workshop, Gaithersburg, Maryland, January 16-18, 1990

Hypertext is most useful as a technology when it is embedded in an application: a paperless technical manual, a notetaker, a specification management system, or any other task domain where it is useful to represent and manipulate the structure of text. We feel that it is important to connect system requirements for hypertext with the situation of use; thus standardization efforts should be directed at enhancing the ability to embed hypertext in heterogeneous applications environments.

This paper addresses a specific application and task environment - using hypertext as a medium for a shared notetaker that will be used in the intelligence community - and how it suggests a protocol-driven approach to integration. The work described in this paper includes an informal work practices study of the task environment, and the development of a functional specification for a hypertext system for notetaking.

From the study and the development of a specification, we postulate that standardization of a multi-tiered system of linking protocols will help address the closed-world problem that we have encountered in NoteCards and many of the other second-generation hypertext systems without specifying rigid standards for applications that want to share information to a greater or lesser extent with a hypertext substrate. Such a system of protocols can be based in part on existing work on hypertext exchange and hypertext reference models.

First we will briefly describe the task environment and present an informal model of the task. Then we will go on to describe linking and anchoring requirements in support of this task. Finally, we will argue that a multi-tiered system of linking protocols will not only meet the needs that we have already identified, but will be adaptable as the environment changes and will facilitate information sharing. It is this set of protocols that we propose should be standardized based on negotiations between applications developers and the hypertext community.

An informal model of analytic activities

The specification we developed describes a hypertext system to support intelligence analysts in their notetaking and other sense-making activities. We based the specification on requirements derived during the course of an informal work practices study that we conducted at the user site, coupled with our previous understanding of the idea processing task (see [Halasz et al. 1987], [Trigg et al. 1986], and [Trigg et al. 1987] for discussions of various aspects of idea processing in NoteCards).

The analysts we studied work in a rich, complex environment of systems and information sources. From these sources they gather information, mostly by scanning the cables they receive through an institutional mail system, or by retrieving information from a variety of on-line resources (including outside information services like Dialog). They read and interpret information they gather, manifesting their interpretation in one of several ways. Sometimes they take notes on what they read or annotate the sources before filing them in their personal on-line or hardcopy file systems; in other cases they reflect their understanding of the material by simply filing source material or organizing it in response to a specific assignment. The product of this interpretation process is usually either a formal written analytic paper, or a shorter (and less formal) article.

Thus, information gathering and retrieval, interpreting sources through notetaking and filing, and authoring reports are all important parts of the analytic task. These processes interact in a variety of ways; notetaking can be driven by information gathering, culling an electronic mail inbox, or it can be driven by the production of a written report. Retrieval needs may be refined in the interpretation process as the analyst tries to make sense of the information at hand, or they may be related directly to a specific assignment. Structures to organize information may also be dictated by either sources or products, or by the internal models of a domain that an analyst has evolved over his or her career. Finally, presentations may be prompted by analytic requirements, or they may be driven by new interpretations that come out of the earlier processes in the flow.

Furthermore, we found that the broader categories of analytic information processing are collaborative or coordinated with people in other organizational roles. Interpretation is often collaborative, sometimes involving telephone conversations, or (less commonly) informal face-to-face meetings. Interpretive collaboration is initiated by three different types of questions: (1) "What do you make of it?" (2) "Do you agree with this (or can you corroborate this)?" and (3) "What are the implications of this?" If the collaboration looks to be fruitful, a draft-passing co-authorship is negotiated between the two analysts, hence starting a presentation-phase collaboration. Coordination occurs in retrieval tasks in two ways: (1) Some members of the analytic work group have specific expertise in retrieval and can help an analyst gather information he or she needs from the institutional or outside sources. (2) Some analysts have specific resources (like their own extensive files); it is a coordinated effort to locate the desired information from those files.

Figure 1 sketches the flow between the categories of analytic activities and shows how they may be conducted in a collaborative setting.

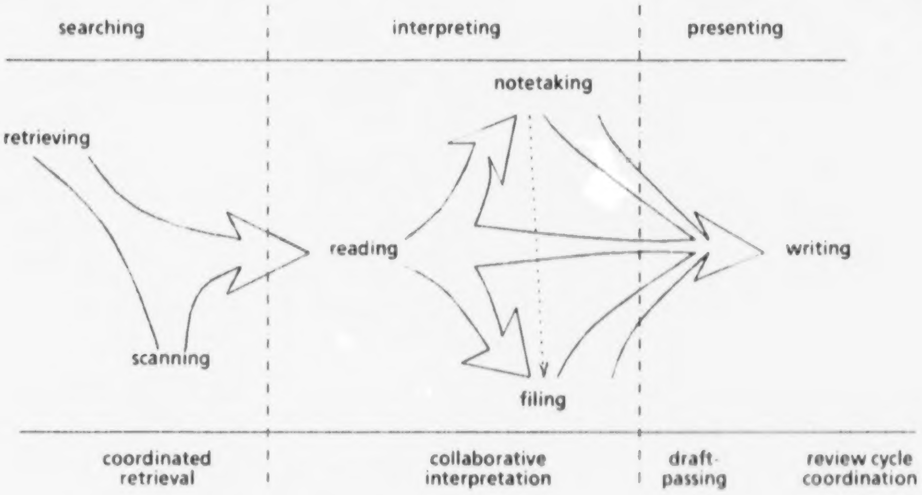


Figure 1. Analytic information processing activities

In order to determine requirements for hypertext in the context of this task environment, it is important to investigate three areas: (1) where the information comes from; (2) the relationship between the kinds of notes analysts take and the information sources; and (3) what use the information is put to after the interpretation is complete. From looking at (1) and (3), we will be able to determine a strategy for integrating hypertext into an applications environment, and from (2), we will understand requirements on linking pieces of information together.

Where information comes from. The analysts we studied use a variety of sources, some currently available on-line or destined to be on-line in the foreseeable future, and others that will continue to be available only in hardcopy forms. Frequently cited anecdotal evidence suggests that only five percent or so of the available data is ever used in analysis; therefore, analysts all feel very strongly about pulling in material from a variety of sources and processing as much of it as possible. It is a widely held belief in the intelligence community that contradictory analytic results stem from the use of different sources, rather than from different interpretations of the same facts.

We have categorized the sources of on-line information that analysts use into four groups: personal files and databases, information from systems maintained by the analyst's working group, information from institutional databases and mail systems, and information maintained external to the organization such as open literature databases. These catagories suggest that there are varying degrees of control that hypertext developers will have over the systems and databases supplying this information. At best - as in the case of personal files and working group databases - the hypertext substrate will be able to represent and display the information at both ends of a link; at worst - the cases where commercial information sources are used - the hypertext substrate will only be able to represent a method for initiating the outside application.

In our study, the most important source of day-to-day on-line information is the institutional mail system that supplies each analyst with cable traffic, filtered by an interest profile. Each analyst described a process of going through the day's institutional mail in a linear sequence and deciding which messages are of interest. Currently, these messages are hardcopied for further processing, mainly highlighting and otherwise marking them up. Therefore, the most prevalent example of where the information comes from falls between the two extremes.

How notes are related to sources. The analysts we studied exhibited a range of notetaking styles. Many of them relied strictly on *annotative* notes; that is, they would make hardcopies of source materials, and mark up the pages. Annotative notes are taken in two different ways. Often, a broad-tipped highlighting pen is used to go over words, sentences, or paragraphs of particular interest. Some analysts have a preference for specific colors when they are doing this type of highlighting annotation. The second annotative style of notetaking involves writing short notes in the margins of the hardcopy. For example, one of the analysts marked things he did not believe to be true, or that he found anomolous; he noted those beliefs in the margins. Annotative notes are closely bound to selected segments of text; in hypertext terms, they rely on access to a portion of the *content* of a node.

We found that the analysts also use *interpretive* notes to record hypotheses, conclusions they have reached, or material they have integrated from several sources. These notes are frequently taken on-line in the text editor; sometimes this style of notetaking involves a significant amount of retyping to associate notes with their sources. Analysts also take interpretive notes that do not refer directly to any source, or that refer to a computational model. Interpretive notes are less tightly bound to individual words or sentences in a document. More often, they refer to a general assimilation of the document's content. Thus they frequently point to what would be represented in hypertext as a *node*.

All of the analysts in our study made some use of *reminding* notes, Post-its or other jottings on paper that serve to jog their memory about things to do (an agenda of subtasks) or portions of procedures to follow (for example, how to log on to a given outside data service, or how to retrieve a piece of information). Reminding notes may be an important way of preserving procedural knowledge. These notes often do not refer directly to a node or its content, but rather how to get to it; they can be thought of as referring to the *link*.

Figure 2 summarizes the three categories of notetaking styles we observed in the work group.

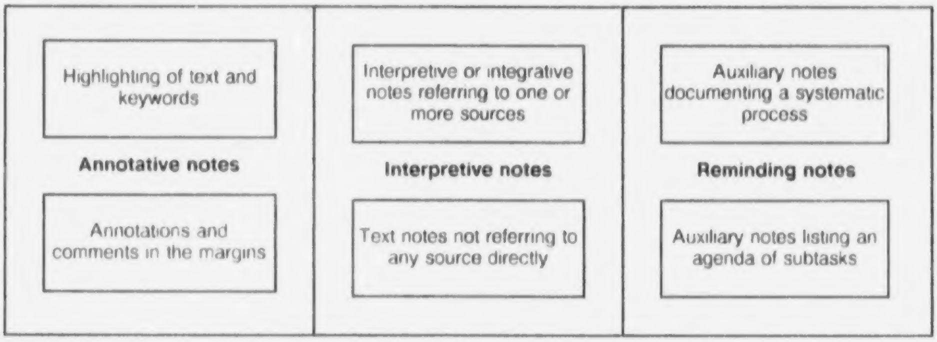

Highlighting of text and keywords
Annotative notes
Annotations and comments in the margins
Interpretive or integrative notes referring to one or more sources
Interpretive notes
Text notes not referring to any source directly
Auxiliary notes documenting a systematic process
Reminding notes
Auxiliary notes listing an agenda of subtasks

Figure 2. Analysis of notetaking styles

How information is used. Information is used two ways: analysts build up personal files and they write analytic reports and short articles, artifacts recognized by the community. This paper will not discuss our findings about how notes and collected information are filed. Instead we will focus on the use of information in analytic products, since one analyst's filing structure is usually opaque to the other analysts. It is difficult for analysts to retrieve information from one another's files, and once an analyst leaves the organization, his or her files quickly deteriorate in value. Thus, in order to make the information useful to anyone else, the analyst must either document this structure or publish any interesting analytic results.

Two kinds of analytic products are supported by the institutional system, formal publications and shorter articles. These analytic products are created by integrating on-line sources and notes, and collections of annotated hardcopy material. Most of the analysts pull out their collection of materials on the desired subject to create a context for writing and to maintain traceability, which is universally cited as an important requirement on (and role for) hypertext. In all cases, the publication of an analytic product, and the subsequent usefulness of the document or article is directly related to the ability to, in hypertext terms, follow its links back to the sources.

Once an analytic product has gone through the coordination cycle, it may be used by low level policy-makers, by various staff members, and by other analysts (sometimes affiliated with different agencies). Analysts expressed a desire for a "lighter weight" analytic product in order to share smaller chunks of analytic results with their community and receive credit for coming up with these results; in hypertext terms, we might think of this as sharing an interpretive layer over a heterogeneous collection of databases.

Linking and anchoring to support of notetaking

From our observations about notetaking in the analytic process, we have derived a set of requirements on links, how they are anchored, and what this implies about an integration strategy.

Links are named, typed, and have direction. Because we expect a variety of relationships between nodes (for example, an analyst might want to specify relationships like *source*, *supports*, or *refutes*), links must be *named*. Furthermore, since we expect links to have different characteristics, links must have *types*, so that a behavior can be associated with the named link. In NoteCards, we have found that the ability to specify the *directionality* of a relationship to be somewhat difficult for users; however, we still feel that representation of the direction of a link may be useful for expressing dependencies.

Links are n-ary. For a hypertext notetaker, n-ary links are important for representing the relationships implied by what we have called interpretive notes. An interpretive note can integrate or synthesize the information in more than one source; hence, the link from the note to the source would require multiple endpoints to accurately represent what is going on in the notetaking process. Figure 3 illustrates an n-ary link example. In this example, Note #1 integrates material from the highlighted portion of Source A and Source B.

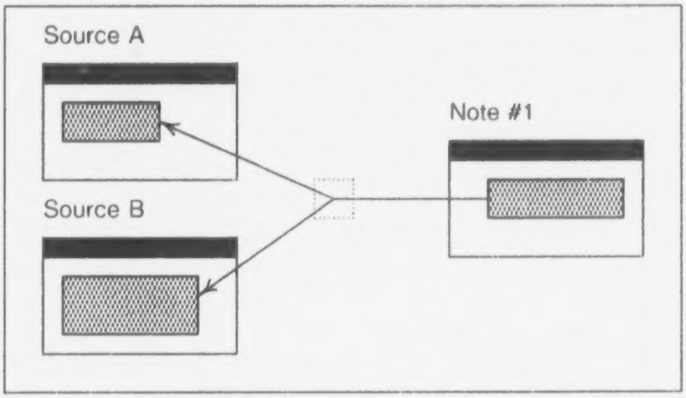


Figure 3. Example of how n-ary links may be used in the notetaker

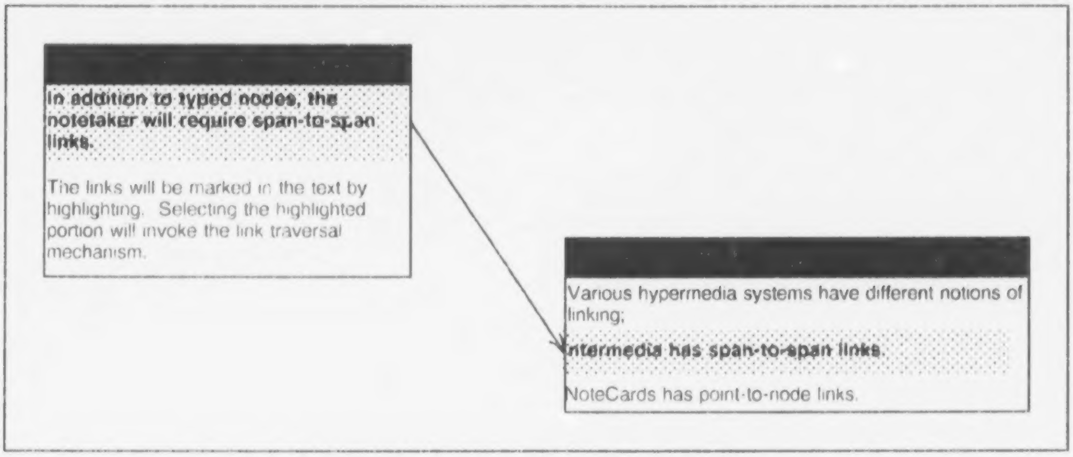
Links can either connect nodes or refer to nodes. There are two different notions of linking in hypermedia systems. Reference links are components within a node that contain a name or address that refers to another node (or a region within another node), or a procedure for retrieving that node; thus a link's destination can be computed at traversal. Reference linking is important in the case where an analyst is performing a query to an external database and wants dynamically computed results.

Connection links are components that connects a node or region within a node with another node or a region within it; the objects at both ends of the link "know" about the link. For the purposes of the notetaker, connections will provide a stronger tie between the information at the source and the annotative or interpretive note at the other end of the link.

Links can be anchored in a span of text. A link anchor is the span within a node corresponding to the endpoint of a link. In some hypermedia systems the span may be limited to a single point (eg.

NoteCards [Halasz et al. 1987]) or to the entire node (eg. gIBIS [Conklin & Begeman 1988]). Other anchoring schemes (eg. Intermedia [Garrett et al. 1986]) may allow anchors to encompass arbitrary extents of text (or graphics) within a node.

Analysts' notetaking practices suggest a need for "span-to-span" links, where an arbitrary region or collection of objects can be connected with another arbitrary region or collection of objects as illustrated in Figure 4. Span-to-span linking is important to the notetaker because most source-connected notes that analysts take generally refer to a region of text. Furthermore, it is important to identify which parts of a multi-source note or a document refer to which sources.



In addition to typed nodes, the notetaker will require span-to-span links.

The links will be marked in the text by highlighting. Selecting the highlighted portion will invoke the link traversal mechanism.

Various hypermedia systems have different notions of linking;

Intermedia has span-to-span links.

NoteCards has point-to-node links.

Figure 4. Span-to-span linking

More specifically, span-to-span linking supports the kind of annotative notetaking that we have observed. The anchoring and marking process is similar to the highlighting that analysts use to set apart a region of text. In this case, it is the delimiting of text that is important; a special link type can support this span-to-null link. The ability to include marginalia as annotations depends on using a span-to-node or span-to-span link. See [Catlin et al. 1989] for an example of how span-to-span linking can support annotation.

Links are marked to reflect their properties. Link markers are the method by which the system indicates the presence of a link anchor to the user. What information a link marker displays should reflect its function. Link markers in the notetaker should allow an analyst to detect the presence of a link without requiring extra action (as an annotation can be detected), distinguish the level of integration of the link's destination, and determine the scope of the anchor's span (as highlighting shows scope).

Links can be annotated. Because procedural or reminding notes sometimes refer to links, rather than to nodes, links should have the ability to be annotated. In the case of very shallow linking (where the actual reference is not sufficient to resolve what should be at the other end of the link), link annotation can supplement automated link resolving mechanisms.

Levels of Integration

This set of requirements on links, coupled with the analysts' need to trace notes and finished intelligence back to its sources and their use of a variety of tools in the sense-making process, leads us to a multi-tiered integration scheme. Of the different tools and applications available in the analysts' environment, some will be more amenable to deep integration than others. Furthermore, we have found that the various kinds of notes that analysts take require greater or lesser connection to outside information, and that in some situations, the payoff for deeper integration is large, while in others, shallow integration is all that is necessary.

We have divided integration into three levels, listed in order of depth: (1) data or content based integration; (2) tool or node based integration; and (3) display or window based integration. This list suggests a need for three protocols, which we feel are general to embedding hypertext in a heterogeneous application environment: an anchoring protocol, a linking protocol, and a launching protocol. Figure 5 summarizes the relationship between the protocols and the depth of integration.

DEPTH	PROTOCOLS		
	anchoring	linking	launching
data/ content	●	●	●
tool/ node		●	●
display/ window			●

Figure 5. Relationship between protocols and depth of integration

At the deepest level, integration requires access to the content of a node. Integration at this level implies that applications must obey an anchoring protocol to describe the extent of the anchor within the node, a linking protocol to retrieve nodes from applications outside the notetaker, and a display protocol so the notetaker can present the node in a suitable window. Deep integration makes it possible to treat information from outside the hypertext system the same way as it is treated within the system; thus traversing in a link is the same as it would be were the node maintained by the notetaker.

At the next level of integration, linking is supported so nodes of information from other applications can be included; in this case, the application only needs to implement the linking and display protocols. In this case, traversing a link is a *retrieval* of a piece of information outside the notetaker.

Display-based integration is the most superficial of the three levels. The purpose of display-based integration is to provide access to outside tools; at this superficial level of integration, traversing a link is a *launch* of an application in a window.

Figure 6 shows a hypothetical notetaking situation, where an analyst has taken a note referring to three outside sources, one at each level of integration. The first text span of the note is integrative, and refers to the first two outside nodes; protocols tell the notetaker how to launch each application and retrieve the appropriate node. Because the node from the first application supports anchoring, the extent of the anchor's span is also marked. The note's second span of text refers to the entirety a node in the second application; linking is supported, but anchoring is not, so only the node can be retrieved and displayed. The third span of text in the notetaker's node refers to some portion of the application launched in the third window. Since neither linking nor launching is supported, the application can only be brought up in a window. The annotation on the third link object is the user's procedural note describing how to get the proper information from the third application.

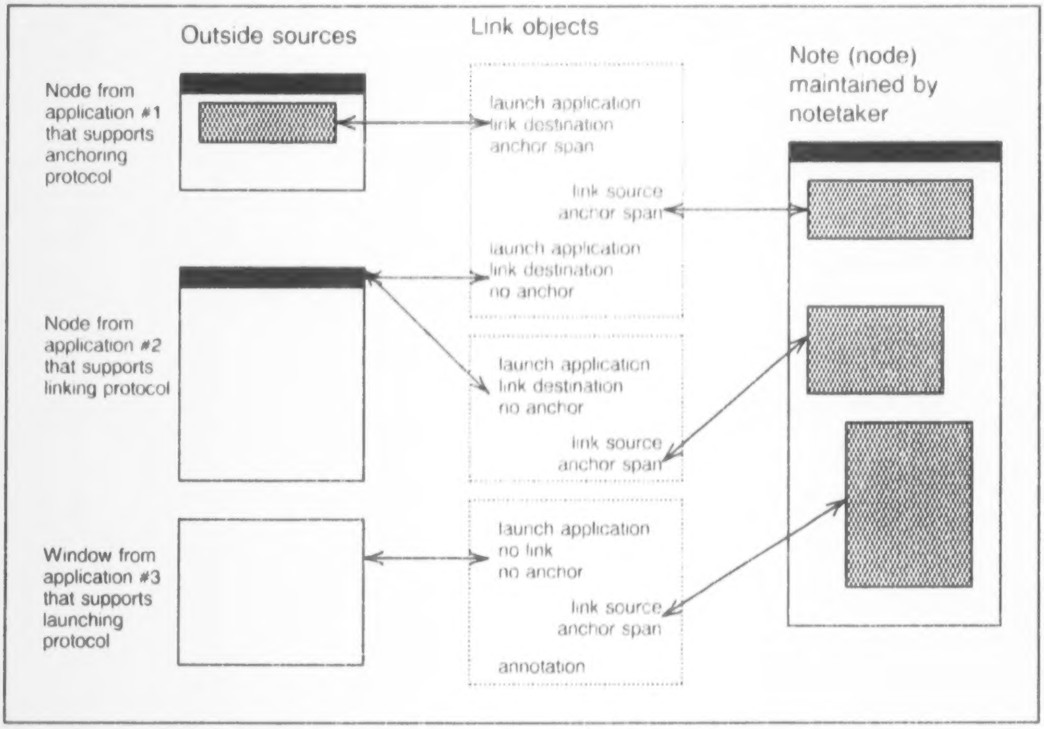


Figure 6. Hypothetical notetaking situation contrasting levels of integration

Defining the three levels of protocol will allow the launching, linking, and anchoring specifications to be expressed and stored in the link objects, and understood by the outside applications to the degree that they support the protocols.

Conclusion

In this paper, we argue that standardization efforts should not only be concerned with a hypertext reference model, but also a multi-tiered system of protocols for integrating information from a heterogeneous applications environment. We make this argument using evidence from a study of a sense-making activity, taking notes in the performance of an intelligence analysis task; we feel that this activity is representative of a wider class of idea processing tasks, and that the applications environment shares many characteristics with other environments where hypertext will provide particular leverage on work involving representing and manipulating the structure of text.

The study we have performed shows that the closed-world assumption at the root of many second-generation hypertext systems limits the ultimate usefulness of those systems, and that future hypertext work must consider at least partially open architectures. Thus creating standards for hypertext necessarily includes developing protocols for integration of outside applications. Our results suggest that three levels of protocols will be useful, an anchoring protocol, a linking protocol, and a launching protocol. These protocols can be closely tied to the reference model adopted by the hypertext community (see [Halasz & Schwartz 1989]) to ensure a common description of what is included in each protocol.

Acknowledgements

I'd like to thank Frank Halasz for some helpful discussions during the development of the notetaker specification.

References

[Catlin et al. 1989] Catlin, T., Bush, P., and Yankelovich, N., "InterNote: Extending a Hypermedia Framework to Support Annotative Collaboration," *Proceedings of Hypertext '89*, Pittsburgh, Pennsylvania, November 5-8, 1989, pp. 365-378.

[Conklin & Begeman 1988] Conklin, J. & Begeman, M., "gIBIS: A Hypertext Tool for Exploratory Policy Discussion," ACM Transactions On Office Information Systems Vol. 6, No. 4, October, 1988, pp. 303-331.

[Garrett et al. 1986] Garrett, L.N., Smith, K.E., and Meyrowitz, N., "Intermedia: Issues, strategies, and tactics in the design of a hypermedia document system," *Proceedings of the Conference on Computer-Supported Cooperative Work*, Austin, Texas, December 3-5, 1986, pp 163-174.

[Halasz et al. 1987] Halasz, F. G., Moran, T. P., Trigg, R. H., "Notecards in a Nutshell," *Proceedings of the ACM CHI + GI Conference*, pp. 45-52, Toronto, 1987.

[Halasz 1988] Halasz, F.G. "Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems," *Communications of the ACM,* Vol. 31, No. 7, July 1988, p. 836-852.

[Halasz & Schwartz 1989] Halasz, F.G. & Schwartz, M., "A Reference Model for Hypertext," *Submitted to the Hypertext Standardization Workshop*, Gaithersburg, Maryland, January 16-18, 1990.

[Trigg et al. 1986] Trigg, R. H., Suchman, L., Halasz, F. G., "Supporting Collaboration in NoteCards," *Proc. of Conference on Computer Supported Cooperative Work*, Austin, Texas, December 3-5, 1986, pp 153-162.

[Trigg et al. 1987] Trigg, R. H., Moran, T. P., Halasz, F. G., "Adaptability and Tailorability in NoteCards," *Human-Computer Interaction - INTERACT '87*, H.-J. Bullinger & B. Shackel (Eds.), Elsevier Science Publishers B.V. (North-Holland), 1987.

10. Newcomb, Steven R. - Explanatory Cover Material for Section 7.2 of X3V1.8M/SD-7

Explanatory Cover Material for Section 7.2 of X3V1.8M/SD-7, Fifth Draft.

Steven R. Newcomb,
Vice Chairman, X3V1.8M, and
Associate Director, Center for Music Research, Florida State University

The mission of the ANSI X3V1.8M Music in Information Processing Standards (MIPS) committee is to develop a Standard Music Description Language (SMDL) to enable interchange of musical documents. The committee has chosen to represent the structure of the information represented by SMDL as a Standard Generalized Markup Language (ISO 8879-1986) Document Type Definition (an "SGML DTD").

In the course of its work (which began in 1986), the MIPS committee developed a general model for the representation of schedules for the execution of events. When it confronted the problem of representing music in several of its normal contexts, such as the interdependently synchronized lighting, staging, and orchestra cues in musical comedy and opera, the MIPS committee developed SGML-based means of representing links within and among documents. These means are what is set forth in the following extract (Section 7.2 ["General Links"] of the fifth draft of X3V1.8M/SD-7 ["Hypermedia/Time-based Document Subset"].

When it became clear that this model would be useful for the representation of the scheduling of non-musical (as well as musical) events multimedia and hypermedia documents, the committee extracted the time model from the other, strictly music-related portions of SMDL, gave the model a name ("HyTime"), and placed it in its own Standing Document, X3V1.8M/SD-7. In the current draft of SMDL, Standard Music Description Language (SMDL) is an application of HyTime. (The rest of SMDL is described in X3V1.8M/SD-8.)

When HyTime's "General Links" facilities were discussed at the NIST Hypertext Workshop, it turned out that the Dexter, Intermedia, and HyTime models all decomposed the problem of document addressing in much the same way, although their jargon was dissimilar. The "Room 705 Ad Hoc Group" (Ed Fox, Steve Newcomb, Tim Oren, and Victor Riley) succeeded in showing how the "anchor" concept in the three models could be merged. It is anticipated that the NIST Hypertext Workshop will have significant impact on succeeding drafts of HyTime.

X3V1.8M/SD-7 Fifth Draft

August 11, 1989

X3V1.8M/SD-7 Journal of Development
Standard Music Description Language (SMDL)
Part Two: Hypermedia/Time-based Document Subset (HyTime)

EDITORS:

Charles F. Goldfarb, IBM Almaden Research Laboratory

Alan D. Talbot, New England Digital Corporation

Includes work as of June 22, 1989. Effective through October 31, 1989

7.2 General Links

General links are relationships between documents or parts of documents. The set of potential general links is infinite, so the mechanisms provided by HyTime are extensible by users and applications.

> **Note:** The term "general link" is used in preference to the unqualified term "link" to avoid confusion with the SGML link feature. However, there is no problem in using "link" with more restrictive qualifying adjectives, as in "hypertext link," or with no qualifiers when the context is clear.

Some forms of general link occur in all documents, not just those intended for hypertext and hypermedia access. Those forms are represented by inherent SGML functions, so HyTime does not need to address them.

Note: Some examples are:

- Links that associate a semantic role (such as "paragraph" or "heading") with an element are represented in SGML by generic identifiers.
- Other links that associate a property with an element (rather than associating two elements with one another) are usually represented in SGML by attributes.

> **Note:** (*"EDITOR"*) We may want a specialized link element nonetheless, for those cases in which the document cannot be modified to add an attribute.

- Links that specify layout or typography, or other processing of a document, are represented by the SGML link feature.
- Links between the logical structure of the document and physical storage are expressed by the SGML entity mechanism, which includes the ability for a user to segment and link a document physically on whatever boundaries he requires.

The following forms of general link are supported by HyTime, either via inherent SGML mechanisms, or by elements and attributes defined in this Standard. (The list is derived from "A Tentative Listing of Some Linktypes" on pp.4/52-4/55 of Ted Nelson's *Literary Machines*, Edition 87.1)

> **Note:** (**EDITOR**) This list represents one view of the requirements for general link support, and as such provides an initial touchstone against which to evaluate the language design. It is provided merely as a starting point, and it is expected that others will suggest additions and modifications to both the list and the design.

a) metalinks
  title
  author
  author (external claim)
  document supersession link
b) ordinary text links for sequential documents
  correction link
  comment link
  counterpart link
  translation link
  heading link
  paragraph link
  inclusion
  quote-link (annotated inclusion)
  layout, typography, epigraphy links
  footnote link
c) hypertext links
  vanilla jump-link
  modal jump-links
  suggested-threading links
  expansion links
d) literary links
  citation link
  alternative-version link
  comment document
  certification links
  mail link

Links can also solve the unique structural problems of interactive multimedia documents, such as instructional materials. For example, when the normal sequence of elements is interrupted by a user response, links in audio material could indicate suitable jumps to graceful endings.

In HyTime, general links all consist of one or more "link ends" (Nelson calls them "end sets"), together with a description of the purpose of the link (the "link type"). A general link also has an associated "link term" that an application displays as a "button" from which the link can be accessed. In character text, the link term is a word or phrase that is the subject of the link, and the "button" is usually the link term in a highlighted font. In other data, the link term is a location (for example, a coordinate in a displayed image), and the button might be a cursor that changes shape when it is over the link term location.

> **Note:** (**EDITOR**) Do we need the potential for a link term at each link end?

HyTime includes four element types that represent general links:

- The independent link is the most flexible. It can have any number of link ends and they can be in any documents, even those to which there is no write access.
- The contextual link has only two link ends, one of which is at the location of the contextual link element.
- The excerpt is a special form of contextual link that is used for including portions of other documents, with or without acknowledgment.
- The location reference is a special form of contextual link that is used for automatic cross-referencing within a document.

7.2.1 Independent Link

The element **independent link** (*ilink*) represents a general link whose link ends are independent of the ilink element itself. The content of the ilink element, if present, is the link term.

An independent link occurs, as its name implies, out of the normal context of the document. Its location need have no connection with the location of its link ends.

> **Note:** An ilink can be used in situations where it is not possible to modify the link end locations. If one of the link ends can be modified, it may be more convenient to use a contextual link (see 7.2.2).

The attribute **linkends** (*link ends*) identifies one or more locations that are the subject of the link. Each can be a document location, data entity location, or some other element, including another general link. The number of link ends, and their meaning, are a function of the link type, which is determined by the application.

The attribute **independent link type** (*ilinktyp*) identifies the purpose of the link. The possible values are determined by the application.

> **Note:** Uses for independent links include comments and notes by reviewers and collaborative authors, external thesauri and indexes, and identification of various kinds of alternative versions.

The attribute **link term** (*linkterm*) identifies the link term of the link. If not specified, the content of the ilink element is the link term.

The entity *a.ilink* allows additional attributes to be defined.

```
                   <!-- 7.2.1  Independent Link -->
<!ELEMENT ilink    -- Independent link: independent of its location (included) --
                   - o  ANY >
<!ENTITY % a.ilink " " -- User-defined independent link attributes -- >
<!ATTLIST ilink    id       -- Used when this ilink is linked to --
                            ID      #IMPLIED
                   linkends -- Ends of link: element, docloc, or entloc --
                            IDREFS  #REQUIRED
                   ilinktyp -- Purpose of link (application-defined) --
                            CDATA   #IMPLIED  -- Default: implied by GI --
                   linkterm -- Index term or "button" location --
                            IDREF   #CONREF   -- Default: content of ilink --
                   %a.ilink; >
```

7.2.2 Contextual Link

The element **contextual link** (*clink*) represents a general link with two link ends. One of the link ends is the content of the contextual link element, which must be valid in the context in which the clink element occurs. The content can be entity if the link end is simply a point in the text, rather than a span of a character string.

A contextual link occurs, as its name implies, in context at exactly the location of one link end. The content of the contextual link element, if it is not empty, is the link term as well as a link end. It is also treated as part of the content of the containing element, just as if there were no clink tags around it.

> **Note:** A clink can be used only if the link has only two ends and one of them can be modified to incorporate the clink tags. In other cases, the independent link can be used (see 7.2.1).

The attribute **linkend** (*link end*) identifies the other end of the link. It can be a document location, data entity location, or some other element, including another general link. The meaning of the link end is a function of the link type, which is determined by the application.

The attribute **contextual link type** (*clinktyp*) identifies the purpose of the link. The possible values are determined by the application.

> **Note:** Uses for contextual links include various forms of hypertext links and alternative access paths through a document.

The attribute **automatic return** (*return*) indicates whether processing of the document returns automatically to the end of the clink after processing the link end.

The entity *a.clink* allows additional attributes to be defined.

```
                    <!-- 7.2.2  Contextual Link -->
<!ELEMENT clink     -- Contextual link: nested subelement of its parent --
                    - o  ANY >
<!ENTITY % a.clink " " -- User-defined contextual link attributes -- >
<!ATTLIST clink     id        -- Used when this clink is linked to --
                              ID        #IMPLIED
                    linkend   -- Other end of link: element, docloc, or entloc --
                              IDREF     #REQUIRED
                    clinktyp  -- Purpose of link (application-defined) --
                              CDATA     #REQUIRED
                    return    -- Automatic return at end of linkto element --
                              (return|noreturn) noreturn
                    %a.clink; >
```

7.2.3 Excerpt

The SGML external entity reference is the normal vehicle for including text from one document within another. Such inclusion is transparent, in the sense that if the included material is itself represented in SGML, an SGML parser will deal with it without advising the application program. Therefore, if an application wishes to acknowledge that certain material is included from other documents, an additional construct is required.

The element **excerpt** (*excerpt*) is a type of contextual link that identifies a portion of another document (the "excerpt source") that is included in this one. In other words, the excerpt source replaces the excerpt element. The included text must be valid in the context in which the excerpt element occurs.

The attribute **quote** (*quote*) indicates whether the existence of the inclusion is made evident to the reader of this document.

The attribute **excerpt source** (*xsource*) identifies the location of the text to be included. It points to a document location or data entity location element that describes a location in a document other than the one in which this excerpt element occurs.

The attribute **acknowledgment** (*ack*) identifies the location of acknowledgment data for the included material, such as a copyright notice. The acknowledgment can be in any notation suitable for use in conjunction with the included material; for example, an image that can be overlayed on an included video clip.

```
                    <!-- 7.2.3 Excerpt -->
<!ELEMENT excerpt  -- Part of another document included in this one --
                   - O      EMPTY      >
<!ATTLIST excerpt  id       ID         #IMPLIED
                   xsource  IDREF      #REQUIRED
                   quote    -- Reveal existence of excerpt --
                            (quote|noquote) noquote -- Default: conceal --
                   ack      -- Acknowledgment text --
                            IDREF      #IMPLIED >
```

7.2.4 Location Reference

Applications that use HyTime will frequently define specialized link elements for cross-references to headings, footnotes, and figures. When a document is presented, the reference elements are replaced by the heading text, footnote numbers, or figure captions of the elements to which they refer. The location reference element, in conjunction with the location elements defined later, offers a generalized mechanism for such cross-references.

The element **location reference** (*locref*) is a form of contextual link whose other link end is a location element. An application will normally process a location reference by replacing it with data that is derived from (but is not necessarily identical to) the content of the link end.

> **Note:** A location reference therefore differs signficantly from an entity reference: the latter is an SGML construct whose behavior is defined precisely by ISO 8879, while the behavior of a location reference is entirely application-dependent.

```
                   <!-- 7.2.4 Location Reference -->
<!ELEMENT locref   -- Reference to a location element --
                   - O      EMPTY      >
<!ATTLIST locref   id       ID         #IMPLIED
                   idr      IDREF      #REQUIRED >
```

7.2.5 Locations

A general link must refer to one or more locations in documents. SGML provides two inherent constructs for identifying locations:

a) A unique identifier ("ID") attribute, which identifies a complete element in the same document as the reference to it.

b) An entity name, which identifies a complete entity (frequently data without SGML markup) in the same document from which it is referenced.

These constructs are insufficient by themselves for general links, because the link ends of a general link could be outside the document in which the link occurs, or they could constitute only a portion of a data entity or element. For these reasons, HyTime supplements these constructs with several "location" elements that can be used separately and in combination to represent the following locations:

a) In a data entity, a point or a span of data, either:

   1) in terms of a data content notation (e.g., a video frame number, a coordinate in space, an offset in time); or
   2) in terms of the uninterpreted characters.

b) In an SGML document or subdocument entity, either:

   1) the entire document or subdocument; or
   2) some identified element within it; or
   3) some data location within the identified element (interpreted or uninterpreted).

   Note: (**EDITOR**) In the next edition, the element location facility will be extended to address a span from one element location to another.

7.2.5.1 Data Entity Location

The element **data entity location** (*entloc*) identifies a portion of a data entity. The data could be "character set data," or it could be "notation data," which must be interpreted according to a particular data content notation. The portion could be a single point, or a span of data between two points.

The attribute **data entity name** (*dataent*) identifies the data entity to which the data entity location refers. If not specified, the data entity is the same as that of the previous entloc element.

```
                 <!-- 7.2.5.1 Data Entity Location -->
<!ELEMENT entloc    -- Identifies a portion of a data entity --
                    - 0      (cdloc | ndloc) >
<!ATTLIST entloc    id       ID       #REQUIRED
                    dataent  ENTITY   #CURRENT -- Default: previous entloc -->
```

Character Set Data Location

The element **character set data location** (*cdloc*) defines a single point in character set data, or a span of data between two such points.

The element **character set data point** (*cdpoint*) defines a point in character set data. The point is represented as an integer offset from the first character in the data. A value of 0 refers to the point prior to the first character, except when only one cdpoint is specified in a cdloc, in which case it refers to the point after the last character.

Only characters that an SGML parser passes to an application are counted (for example, a record end after a start-tag is not normally treated as a data character).

```
          <!-- 7.2.4.1.1 Character Set Data Location -->
<!ELEMENT cdloc     -- Character set data location --
                    - 0     (cdpoint, cdpoint?) >
<!ELEMENT cdpoint   -- Character set data point --
                    -- Offset from first significant character --
                    -- 0 = before first char (after last if only one cdpoint) --
                    0 0     (#PCDATA) >
```

Notation Data Location

The element **notation data location** (*ndloc*) defines a point or a span between points in data that is subject to interpretation by a data content notation. The representation of the point or span is not defined by this standard; it depends upon the notation in which the data itself is represented.

In HyTime applications, the data would normally represent occurrences in space, time, or both, so a notation data location would consist of offsets on a visual coordinate system, and/or elapsed time values. Some notations also provide the ability to "label" items for identification. In such cases, a notation data location could refer to such labels.

The attribute **snap** (*snap*) indicates whether the specified location should be adjusted to conform to alignment or synchronization points in the data. The specified location can be "snapped" to the nearest, next previous, or next following alignment point, or not at all.

> **Note:** Graphics representations commonly have an associated "grid" to which objects can be "snapped" in order to assure alignment and/or a minimum resolution. Similarly, representations with an internal time bases frequently include synchronization points, such as frame markers in SMPTE encoding of movies and video.
>
> **Note:** (**EDITOR**) It may be possible to define a generalized method of referencing space and time locations that would serve for a wide variety of notations. Such a method could be incorporated into HyTime as the definition of an ndloc element. The snap attribute is an example of one possible parameter. Suggestions are invited.

```
          <!-- 7.2.4.1.2 Notation Data Location -->
<!ELEMENT ndloc     -- Notation data location --
                    -- Offset in time or space and duration or size, or label --
                    - 0     (formula) -- Depends on data content notation -->
<!ATTLIST ndloc     snap    -- Specified point is changed to aligned point --
                            (nearest|before|after|none) none >
```

7.2.5.2 Document Location

The element **document location** (*docloc*) identifies a portion of an SGML document by means of an optional element location, and an optional data location within that element. If no element location is specified, the "element" is the entire document. If an element location is specified, but no data location, that complete element is the "document location."

The attribute **document entity** (*docent*) identifies the entity in which the document begins. If omitted, it is the same entity in which the docloc element occurs.

```
                 <!-- 7.2.4.2  Document Location -->
<!ELEMENT docloc   -- Identifies a portion of a document or subdocument --
                   -- Entire document if element location is omitted --
                   -- Entire element if data location is omitted --
                   - O      (elemloc, (cdloc | ndloc)?)? >
<!ATTLIST docloc   id       ID       #REQUIRED
                   docent   ENTITY   #IMPLIED -- Default: this document -->
```

Element Location

The element **element location** (*elemloc*) identifies an element either by a unique name, or by a sequence of "node locations," called a "node path." The element location permits a general link to refer to an element in a different document, or to an element (in any document) that does not have a unique identifier attribute ("ID").

The attribute **element identifier** (*elemid*) is the unique identifier ("ID") attribute of the element whose location is being identified. If the element has no unique identifier, its node path is used instead.

Notes:

a) The attribute elemid is not declared to be an "IDREF" attribute because its value may be an ID from another document. An SGML parser will normally check for the validity and uniqueness of an IDREF, but cannot do so for an ID from another document, as it could conflict with an ID from this document.

b) The keyword "#CONREF" identifies a "content reference attribute." If a value is specified for the attribute, the SGML parser will expect the content to be empty (and vice versa). The application is expected to use the attribute value in some way as a substitute for the data that would ordinarily have been in the content.

```
                  <!-- 7.2.5.2.1  Element Location -->
<!ELEMENT elemloc  -- Identifies an element of a document or subdocument --
                   - O      (nodeloc+) >
<!ATTLIST elemloc  elemid   NAME     #CONREF  -- Default: use node path -->
```

Node Location

The element **node location** (*nodeloc*) identifies the sequential position of an element among its siblings in the tree structure of the document. The node location is an integer greater than zero, and each separate data portion in mixed content is treated like an element when counting.

> **Note:** For example, in a paragraph consisting of some character data followed by a quotation element, and then some more character data, the first character string would have a node location of "1," the quotation a node location of "2," and the second character string a node location of "3."

Any element, including the pseudo-elements containing the data in mixed content, can be identified uniquely by a "node path" consisting of an ordered sequence of the node locations of itself and its ancestors, starting at the root of the document tree.

Note: For example, in a document with the following structure:

```
<core><mmseq><ces><ce><ce></ces></mmseq><baton><tempo><tempo></baton></core>
```

the second tempo element can be identified by the node path:

```
1 2 2
```

An element that is empty or that contains only data (including the pseudo-elements containing data in mixed content) is a leaf of the document tree. Its data does not have a node location, but can be addressed with a data location element.

```
                  <!-- 7.2.5.2.2  Node Location -->
<!ELEMENT nodeloc  -- Node location: integer > 0 (each #PCDATA is one) --
                   - 0        (#PCDATA) >
```

7.2.5.3 Point Location

The element **point location** (*pointloc*) identifies a point in an element so that it can be referenced. Its content, which is optional, can be used by an application to describe the point.

Note: For example, when printing a cross-reference to it.

```
                   <!-- 7.2.5.3  Point Location -->
<!ELEMENT pointloc -- Identifies a point in an element --
                   -- Content can be used by application to describe point --
                   - 0        ANY >
<!ATTLIST pointloc id         ID         #REQUIRED >
```

Toward Open Hypertext: Requirements for Distributed Hypermedia Standards

A Position Paper for the NIST Hypertext Standards Workshop

Tim Oren, Apple Computer

1. Directions for Hypertext Standards

Much discussion of hypertext standards has centered on the transfer of closed, static hypertext document bases among various platforms and organizations. While there is an undoubted need focused on the use of hypertext with optical media and technical documentation, the thesis of this position paper is that any standard based primarily on this limited application will be necessarily flawed.

The original vision of hypertext was a universally shared, dynamic "docuverse" which could be read and written by all users. Although systems short of this grand vision have proven utility, we would not wish to abandon this future or the smaller scale visions of department and enterprise-wide hypertexts. Nelson proposed that one unified backend storage mechanism, "Xanadu," would solve the distributed hypertext problem for all [Nelson 80]. Though the Xanadu system is now advancing toward commercial release, it comes late in the day. There are already established commercial hypertext systems and sizable collections of content which are unlikely to be abandoned.

Hence, if we want the docuverse to become reality, we must build it in the distributed, multivendor computing milieu of today. To bring together the diverse software and hardware systems already existing we will need abstract models of hypertext and ultimately standards based on the models. If this work is to be viable, the results must also reflect technical and market realities, and interaction with other areas such as multimedia and compound documents must be considered. In the remainder of this paper, I examine some of the requirement posed by these constraints, propose design principles for meeting these requirements, and suggest that an open system architecture should be the ultimate goal of hypertext standardization efforts.

2. Technical Conditions

Working in today's computing environment means working with existing networking and file standards. These are characterized by loose connectivity and modest reliability. Not only do LANs and WANs break down, but many connections are deliberately noncontinuous for cost reasons. Remote resources such as servers fail and go offline, often due to crashes that mean reloading earlier data versions. Existing file and device level utilities allow copying and alteration of file and document structures without warning to the applications which rely on them. All existing standard user interface systems are aimed at this level. These utilities are used routinely to remove partial document collections for work at home or transfer to other sites, and to return modified versions to the original system. A hypertext standard for this environment must be robust when faced with a variety of insults to document identity and link integrity.

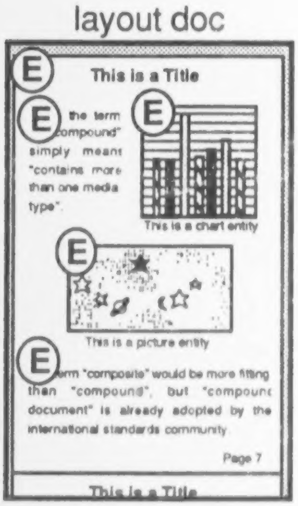

layout doc
E
This is a Title
E
the term "compound" simply means "contains more than one media type".
E
This is a chart entity
E
This is a picture entity
E
term "composite" would be more fitting than "compound", but "compound document" is already adopted by the international standards community.
Page 7
This is a Title

Figure 1. Compound document

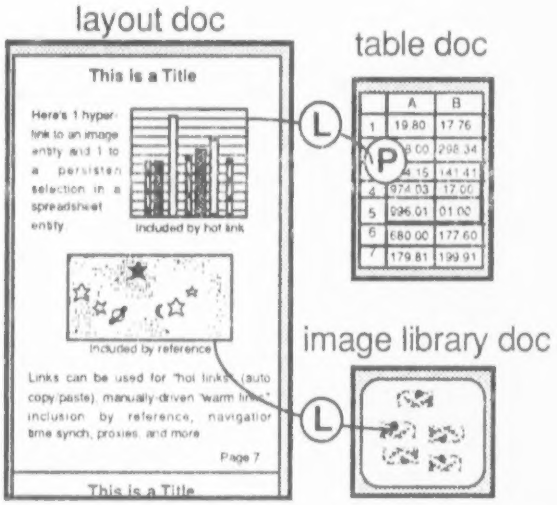

layout doc
This is a Title
Here's 1 hyperlink to an image entity and 1 to a persistent selection in a spreadsheet entity.
Included by hot link
Included by reference
Links can be used for "hot links" (auto copy/paste), manually-driven "warm links", inclusion by reference, navigation, time synch, proxies, and more
Page 7
This is a Title
L
P
L
table doc
A
B
1
19.80
17.76
8.00
298.34
4.15
141.41
4
974.03
17.00
5
996.01
01.00
6
680.00
177.60
7
179.81
199.91
image library doc

Figure 2. Hypermedia document

Activity in hypertext standards interacts with other advanced document models. For instance, figure 1 shows a "compound document" where various text and graphical entities (E) are assembled into a page under the control of a layout specification. However, rather than storing the compound document as a single file, it might be realized as shown in figure 2. Here, a hypertext substrate is used to implement a compound document: the graphic entities are placed using links (L) to persistent selections (P) within other files.

Links can encode dynamics and constraints as well as static information. In figure 2, the upper link specifies the transformation of the linked data into a graph. In figure 3, links are used to specify synchronization information for pieces of dynamic media. Finally, as suggested in figure 4, the rise of object oriented software may make possible "component documents" where each entity may be edited in place by software modules selected at runtime by the user. Implementing a component software system will require a standard data storage substrate very similar to hypertext which vendors of individual components can use and extend.

Because these issues and applications all interlock, it is not possible to restrict a discussion of hypertext standards to static text alone or to particular document models. A standard arrived at in this fashion will suffer one of two fates. At best, it will create a "golden ghetto" where a class of hypertext applications may live, but without connecting to other media types or document models. At worst, it may coopt and prevent progress in these areas.

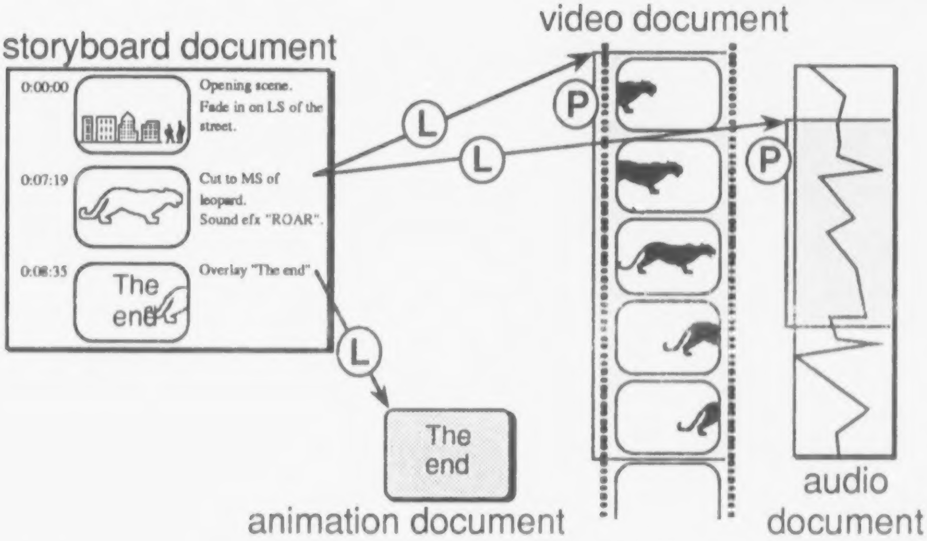
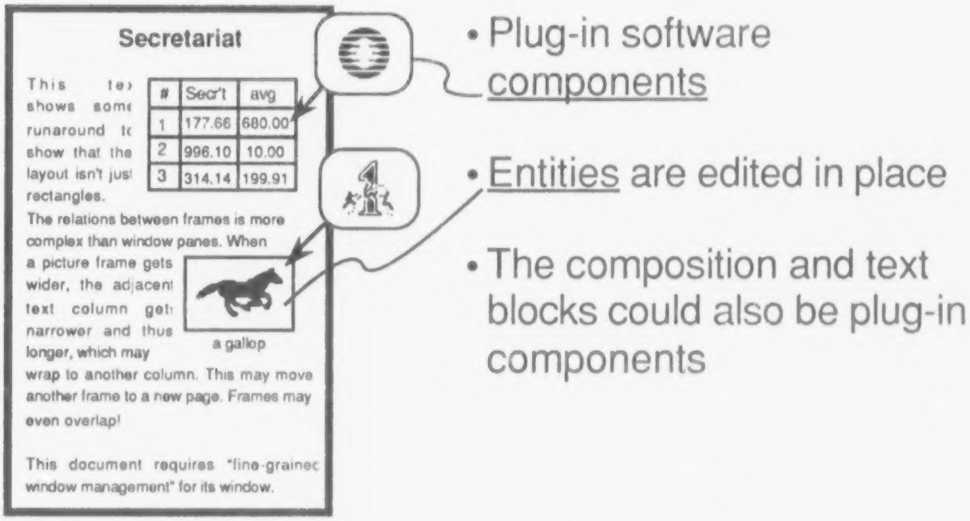


Figure 3. Multimedia documents



Secretariat

This te
shows some
runaround t
show that the
layout isn't jus
rectangles.

#	Sec't	avg
1	177.66	680.00
2	996.10	10.00
3	314.14	199.91

The relations between frames is more complex than window panes. When a picture frame gets wider, the adjacent text column gets narrower and thus longer, which may wrap to another column. This may move another frame to a new page. Frames may even overlap!

a gallop

This document requires "fine-grainec window management" for its window.

- Plug-in software components
- Entities are edited in place
- The composition and text blocks could also be plug-in components

Figure 4. Component documents

In these examples the objects are not exclusively text. They include static bit map graphics and object graphics, dynamic animation, sound and video. Each of these data types represents a corresponding discipline and standards effort. A hypertext standard which restricts itself to text alone is crippled at the beginning. One which attempts to reinvent standards for each constituent media type would create a ghetto effect, and might be simply impractical given the effort required. It would seem that a hypertext standard must find a way to embrace existing media type standards with a minimum of modification. In the remainder of this discussion, I use hypertext in its most general sense, to indicate the scheme for linking all data types, not just text.

Hypertext as a functioning discipline is quite young, and disagreement and lack of understanding of systems architecture and application needs is still rife. There is controversy at even the fundamental level of linking method and storage organization. Various systems implement links as separate webs or within documents, and represent them abstractly or procedurally. A recent panel on system architecture makes it clear that there is still substantial change, with many systems seeking to adapt the better features of the other approaches [Halasz 89]. It is also clear that the diversity of systems is not gratuitous variation, but has occurred because of real differences in the intended applications and audiences. No "one right way" to do hypertext has emerged.

Above the storage level, diversity increases further. Labelled links are used diversely, to represent constraints, timing, inferencing and rhetorical information for the use of both the browser and the software. User interfaces to large, interlinked data stores are an area of active and fruitful research. More complex architectural issues such as versioning and searchability are just beginning to be explored. Again, the various approaches and progress have been largely driven by the needs of particular applications.

Attempts to standardize in a discipline in such flux must take account of the diversity of approaches if they are not to cripple progress. To the greatest extent possible, formalisms must embrace the diversity of architectures and applications rather than being exclusive or prescriptive.

3. Market Conditions

A standard must consider prevailing market conditions to be effective. In the case of distributed hypertext, the installed base of machines on networks is characterized by wide diversity of vendor, architecture, and hypertext software. Significant hypertext systems run on Macintoshes, IBM PCs and PS/2s, Sun, DEC and other Unix equipment, interconnected with a variety of LAN architectures, many also connected to long haul networks such as Bitnet or Milnet. Hypertext software is provided by both hardware vendors (HyperCard, Sunlink) and independent software vendors (KMS, HyperTIES, Guide). Initial market penetration of hypertext technology is occurring in the areas of in-house and external technical documentation and distribution of multimedia content, particularly on optical discs. Substantial commercial and academic efforts are underway to introduce hypertext as a mechanism for collaborative work in the computing environment.

Given this diversity of platforms, the resemblance of distributed hypertext to the open systems efforts undertaken in networking and structured databases is obvious. The existing vendors, applications, and users will not be dislodged by either a proprietary specification such as Xanadu or a public standard. A successful effort must coopt existing users by extending their reach onto other platforms. It should become possible to, for example, read nodes within HyperCard without being necessarily aware that they reside in a remote database created in HyperTIES.

The technical issue of non-textual data also has a market component. Not only do standards for various data types evolve separately, but the markets for the underlying technology in hardware and software progress at their own speed. Of particular importance, there is often a succession of dominant applications within a media type. For instance, on the Macintosh, MacPaint was surpassed in turn by SuperPaint and PixelPaint. A standard must accommodate this process in two ways. First, it must not bind data tightly to its creating application, in order that the user may replace it with another at a later date. Second, the standard must be extensible, to allow vendors to compete on features without being required to abandon the standard.

Another market phenomenon is the decline of the so-called "integrated application." The required feature set within each data type has become so large that a project or product which attempts to do all becomes impractical. Integrated applications linger only at the novice level. Much integration is now done by cut-and-paste or data piping facilities at the operating system level.

Hypertext may be viewed as the next logical evolution of integrated applications, with the ability to freely browse between all data types. Given the issues outlined above, it follows that the hypertext facility will need to be implemented at the system level to be effective. A successful standards effort must then include platform vendors and provide a mechanism for their joint efforts.

The hypertext market is quite young. Many of the software vendors are startup ventures and are thin on capital and engineering resources. A successful standard must address this problem by making implementations available to such developers at very low cost. Failure to do this would confine use of the standard to high-end markets where firms and clients can afford the engineering overhead to implement the standard. It would also cut the standard off from the most innovative sector of the software market. Even a low cost standard must present convincing advantages in integration, power, and room for growth if developers are to give up proprietary schemes of data storage.

4. Design Principles for a Hypertext Standard

What principles can be deduced from these technical and market constraints? First, a standards effort must start with the creation of an abstract model of hypertext which is as inclusive as possible. Because many existing hypertext systems were tightly driven by application scenarios, this means looking at a variety of user communities needs. Particularly, building any system architecture driven by the needs of one application area into a standard would be inadvisable. The work of the Dexter Hypertext Model is a useful precedent in this area [Halasz 90].

Any standard must be portable to the greatest extent, not dependent on particular processor, display, network, or peripheral architectures. Portability will allow the greatest degree of interoperability in the current computing environment, and guarantee survivability onto succeeding generations of technology.

Given the need to incorporate existing data type standards and allow the implementing software to evolve independently, a hypertext standard must support modularity. Data items may be incorporated by reference to an existing file as well as by inclusion within a standard form hypertext. Extensions to existing standards to incorporate hypertext features should be minimal.

A hypertext standard must be extensible to support the rapid evolution of both data type specific software and notions of usage of links. Any typing mechanism built into the hypertext definition must be open to extension. Methods must be provided for superseding one representation of a data element with another without disrupting the entire hypertext. Facilities must be provided for incorporating proprietary data representations with the facility to point at parallel standard representations.

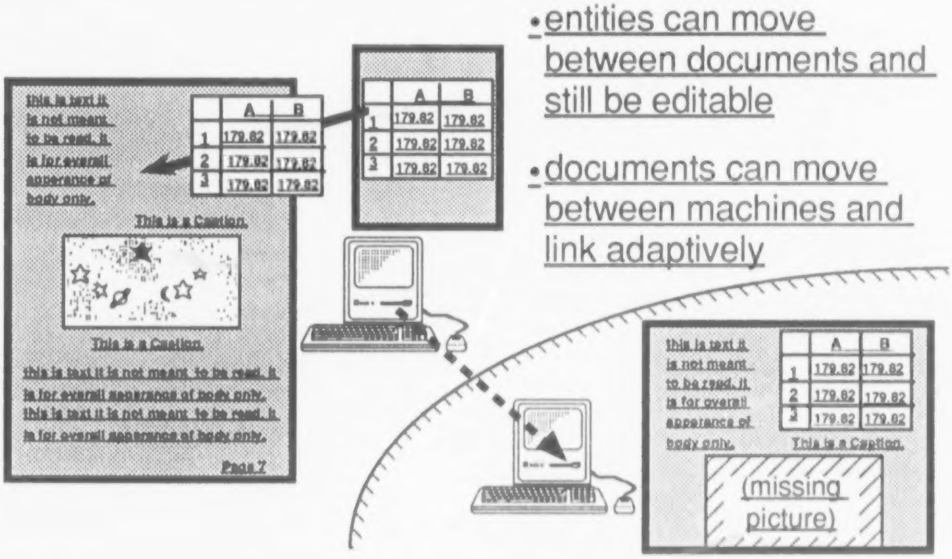

•entities can move between documents and still be editable
•documents can move between machines and link adaptively
A
B
179.82
This is a Caption.
Page 7
(missing picture)

Figure 5. Separability: Moving data around

A principle termed "separability" is important to coexistence with today's file and network systems. This entails, first, a level of data organization called "entities." An entity encapsulates sufficient data and metadata that it may be moved or copied between files without loss of information. For instance, an animation data entity might contain a series of frames, persistent selections for linking, a color lookup table (CLUT), and a description of the required screen resolution and depth and processor resources. This could be moved in its entirety, while copying the frames alone would lose information as they were moved out of context. Figure 5 illustrates this concept, as well as the related feature that entities must be robust in the face of missing linked data. In the partial hypertext extracted to a remote machine the library image is missing, but sufficient layout information remains to block out its location and allow work to proceed.

Separability must be supported with identity and inspectability. A robust identity mechanism allows an implementing system to detect if a referenced entity is missing or present in duplicate. Note that identity may be separated from the particular mechanism which a system uses to find the referenced entity. Various implementations might keep merged databases of entity identity vs. location, or resolve references using heuristic mechanisms peculiar to a platform.

Inspectability means that the interdependencies of entities must be apparent to a utility which understands the linkage standards only, and has no knowledge of the internal structure of data entities. Such a naive utility may then copy or move portions of the hypertext without a need for extensions as new entity types are added.

To allow room for the evolution of hypertext technology, a layered standard will be necessary. To permit layering, each portion of the standard must be policy neutral. This means that it must allow a wide range of choices in how it is applied by higher layers. For instance, a standard which specified link formats and also required their storage in a single "web" would not be neutral, because it enforces a particular implementation. A policy neutral formulation would specify the format and possibly behavior of links without specifying in what place(s) they must be stored. Policy neutrality also permits the delegation of certain design choices to implementors, and provides degrees of freedom for technical issues with no current solution. These issues include the division of entities and linkage information between files, link typing and usage, searchability and version management. Again, an abstract model is helpful in creating the generalizations needed for policy neutrality.

Standards may be expressed as data formats or as behaviors. A hypertext standard expressed as an explicit data format is probably necessary to support environments where only serial ASCII or binary data is available. This is typical of the bulk transfer of reference hypertexts between machines. However, such a format is poorly adapted for update and search. Neutrality of applications is better provided by standardization at the behavioral level of an application program interface (API). A compliant implementation might simply provide access to the standard serial hypertext form, but would more likely implement a random access or object-oriented filing mechanism adapted for its particular platform. The distributed open hypertext environment is then implemented as peer-to-peer conversations among compliant implementations of the standard.

5. Conclusions

Standards must be approached cautiously in a field as new as hypertext. While we may need interim or experimental specifications for particular application areas, making the exchange of static hypertexts the subject of a standard is undesirable. Decisions which we make will necessarily affect other areas such as multimedia and compound documents. A premature standard could have the effect of ghettoizing a subset of hypertext. The goals of a hypertext standard should be the implementation of the vision of distributed hypertext within an open systems framework.

6. Acknowledgements

This paper is based on extensive discussions with Jerry Morrison and Richard Moore, my colleagues at Apple Computer, and was also influenced by members of the Dexter Hypertext Workshops, particularly Norm Meyrowitz, Randy Trigg, Amy Pearl, Frank Halasz and Mayer Schwartz.

7. Bibliography

[Halasz 89] Halasz, Frank, et. al., "Panel: Confessions — What's Wrong with our Systems," given at Hypertext '89, November 5-8, 1989, Pittsburgh, PA.

[Halasz 90] Halasz, Frank and Mayer Schwartz, "The Dexter Hypertext Model," to be presented at NIST Hypertext Standards Workshop, January 16-18, 1990, Gaithersburg, MD.

[Nelson 80] Nelson, T., "Replacing the printed word: A complete literary system," Proceedings of IFIP Congress 1980, North-Holland, 1013-1023, 1980.

Toward a Reference Model for Hypermedia

H. Van Dyke Parunak
Industrial Technology Institute
P.O. Box 1485
Ann Arbor, MI 48106
(313) 769-4049, van@iti.org
7 December 1989

Abstract

A necessary first step in discussing standardization in a domain is the development of a reference model for that domain, a high-level framework within which specific topics for discussion can be defined and discussed. This paper offers a "straw" version of such a framework as a basis for discussion, and discusses the "standardizability" of various detailed subjects within that framework.

1. Introduction

A reference model is a high-level description of a domain within which discussion of more detailed subjects can be situated. As a mechanism for setting the context of a domain, reference models have been useful in several fields. This section gives examples of other reference models, suggests some of the uses to which they may be put, discusses why a reference model is desirable for hypermedia, and outlines the high-level structure of a proposed reference model for hypermedia.

1.1. Examples of Other Reference Models

Reference models have been proposed in many domains, including telecommunications, factory control architectures, and material handling architectures.

Perhaps the best known reference model is the ISO-OSI seven-layer model for telecommunications.[DAY83] By articulating the various communications functions and defining an ordered relation among them, this model has supported a vigorous and productive standardization effort.

A number of studies have proposed reference models for manufacturing control; [PARU87] provides a useful summary, and [BIEM89, WILL89] are more recent treatments. These studies have been motivated by the growing interest in integrated manufacturing, and the resulting need to relate the various entities in a manufacturing enterprise to one another in a consistent way.

In the domain of material handling, the OSI model has been adopted to define a layered model for the transport of material,[PARU88] and this model has been used as the basis for experimental implementations in our laboratory.

1.2. The Uses of a Reference Model

A reference model is useful for description, standardization, design, and innovation.

It provides a descriptive framework for comparing existing systems in its domain, and in fact is often compiled by surveying existing systems for similarities and differences. It thus provides an underlying ontology of its domain.

By identifying the critical subjects in the domain and showing how they are related to one another, it provides a context for standardization. It facilitates discussion of what is and is not ready for standardization, identifies specific subjects for standards, and calls out where subsystems (and thus the standards that describe them) must interface with one another.

As a high-level analysis of its domain, a reference model guides the designer of a new system in identifying the issues that must be addressed and the broad functions that the system must provide, as well as suggesting the kinds of solutions that have been attempted in the past.

Reference models not only help to mature a field through development of standards and common analyses, but can also foster innovation. At the detailed level, by partitioning the problem, they invite the development of new solutions, showing what has already been tried. At a higher level, they invite creative thinkers to challenge their overall structure and thus introduce new paradigms.

The descriptive and prescriptive functions of a reference model are in natural and unavoidable tension. As a guide to classifying existing systems and as a pointer to needed innovation, a reference model should be as comprehensive as possible, able to embrace any implementation of the domain. As a roadmap for standardization or a guide for designers, it should embody design choices that reflect good practice and sound engineering, and thus be selective. It seems reasonable to expect that reference models will follow a life-cycle that moves from broad and descriptive to selective and prescriptive. While it may be premature to build prescriptive models of hypermedia, it is not at all too early to formulate broad descriptions of the underlying technologies, descriptions that through time can evolve into more selective models.

1.3. Why a Reference Model for Hypermedia?

A reference model for hypermedia is desirable not only for helping the technology to mature, but also for fostering its development as a distributed tool.

Every worker in a domain has an individual "reference model" of that domain within which various contributions to the field are implicitly classified and assessed. A textbook in a domain is essentially an instantiation of such a model, and helps newcomers to the domain to put in place a mental framework within which to operate. The rapid growth of interest in hypermedia makes this educational service particularly desirable in the case of hypermedia. However, if this were the only motive, it is questionable whether a joint activity to develop such a model would be justified.

The need for a jointly developed model arises from the potential of hypermedia as a distributed technology. Hypermedia is distributed in at least two ways. First, it has proven to be a useful medium for managing the collaboration of teams of workers.[CONK87, HALA87] Thus it is often implemented as a distributed application, with the resulting need for standards to insure that the various components of such an application are consistent with one another and can be maintained in a modular fashion. This motive for standardization becomes especially strong when the components are not operating in a homogeneous environment. Second, the information that is linked together in a hypermedia system is often distributed in the sense of being of differing types and origins. The ability of a hypermedia system to access generic materials without expensive recoding and preprocessing will depend on the rapid development and broad dissemination of standards for the production and encoding of machine-readable information.

1.4. A Possible High-Level Structure

The reference model sketched in this paper is described from three perspectives: the functional elements of a hypermedia system, implementation concerns, and interface issues. We will outline the main elements to be considered in each of these areas, and also suggest the applicability of standards to each area.

2. Elements of Hypermedia

The two basic elements of a hypermedia system are nodes of information and links that join them together. In addition, recent research suggests that the usability of hypermedia depends on the disciplined use of structured composites of nodes and links as higher-order entities.

2.1. Nodes

The nodes of a hyperbase are the units of information that it assembles together and among which it provides ready movement. The nodes in a system can be described from the perspective of their contents, their typing, and their structure.

2.1.1. Node Contents

The very name "hypertext" suggests that virtually every hypermedia system can present information in the form of text. Most implementations support some form of graphic display as well. Animation, video, and audio are less common but have been demonstrated. [BIEB89] suggests generalizing the notion of a node to "any information item about which the system can reason." Such a definition permits a node to be executable code that is invoked when the link leading to it is traversed, thus leading to any conceivable kind of computer operation. In fact, some early antecedents of hypertext were menu systems, in which all leaf nodes were of this sort.

As long as nodes are treated as atoms, there is no difficulty with such a variety of node contents. For many purposes, one must define locations within nodes, either as destinations or as origins for a link. The mechanisms for such definition are highly dependent on node contents. For example:

- Because text is one-dimensional, location in a textual node is conveniently defined on the basis of characters.
- In graphical nodes, location is defined two-dimensionally on the basis of pixels.
- Animation and video invite the same pixel-based definition of location as does graphics, but there is an additional time dimension.
- Location in an audio node is most readily defined temporally.
- In a node consisting of executable code, the instruction counter is a reasonable measure of location. If the node processes user input, location can be defined in terms of the possible user trajectories through the program.

2.1.2. Node Typing

In addition to different contents, nodes may also have different types. Node typing is most often important in the context of typed links. For instance, in gIBIS, a *Supports* link can only appear between a node of type *Argument* and one of type *Position.*[CONK87] Together with link typing, node typing permits the definition of a grammar or rhetoric over a hyperbase, and greatly facilitates user navigation and automatic information retrieval.

2.1.3. Node Structure

The measures of location defined above for nodes of differing contents are sometimes too primitive for convenient use. For example, one can define words or sentences in a textual node, buttons or sliders in a graphical node, musical phrases in an audio node, or positions in a user trajectory in an executable node, hiding the corresponding characters, pixels, time intervals, or instruction counts as implementation details. Then links can originate or terminate at these higher-order objects. Consistent definition of such higher-order objects and their mappings to lower-order entities offer a good opportunity for standardization.

2.2. Links

A discussion of links in a hypermedia system requires definition of directionality, topology, types, anchors, and modes.

2.2.1. Link Directionality

A link is directional if its ends are differentiated in some way from one another. Often, the mechanism for traversing a directional link in one direction is different from that used in the other direction. For instance, links in Intermedia are not directional. The same icon marks both ends of the link, and the same operation traverses it in both directions. In HyperTies, links are directional, and the backward direction is usually only accessible if one has already traversed the link in the forward direction. Cognitively, directional links can be a valuable aid to navigation in a hyperbase.[PARU89]

2.2.2. Link Topology

Current systems typically do not constrain the overall topology that links can form, but user navigation depends critically on this topology, and there are strong cognitive motives for disallowing arbitrary topologies.[PARU89] The number of possible topologies is countably infinite, but important major classes are linear, hierarchical, hypercube, and DAG.

2.2.3. Link Types

By defining various types of links (and typically correlating them with typed nodes), we can enrich the rhetorical capabilities of a hyperbase, as discussed above under "Node Types."

2.2.4. Link Anchors

The anchors, or endpoints, of a link are its origin and its destination. The destination of a link can either be a node as an atomic unit, or some entity contained within the node. In the case of a structured node, this entity will be some element of the structure. In the case of an unstructured node, this entity will be either a point or a region defined by whatever measure of location is appropriate to the node's contents.

If links are constrained to originate with nodes as atomic units, the resulting hyperbase will have a linear topology, which forfeits the more interesting features of hypermedia. Thus at least the origins of links are some element within a structured node or some location or region within an unstructured node.

2.2.5. Link Modes

The simplest form of a link is a fixed connection between two anchors (either nodes or entities within nodes). The order of processing a link is usually select-traverse-display. Both the form and the processing of a link can be expanded [BIEB89]; a link can be virtual (computed at run-time) rather than fixed, and inferencing can be added both before and after link traversal. Such additional inferencing can be used to implement such modes of linking as warm links (in which users can push or pull data over a link) and hot links (in which data modified at one end of the link is automatically updated on the other end).[CATL89]

2.3. Composites

There has been a growing realization among workers in hypermedia that usable hyperbases require the ability to manipulate composite entities: entities that are larger than, and made up of, individual nodes and links.[HALA87] Such composites can be defined either rhetorically or topologically.

Paths [ZELL89] are a simple example of a topological composite. A bare network of links and nodes is well-suited to random browsing, but many applications of hypermedia presuppose a basic trajectory through the hyperbase, with the rest of the material available as needed. Paths support such applications by giving writers a way to define a backbone that readers should follow, and to which they can readily return after any digressions. Topologically, the path imposes a linear topology on a much more complicated network, thus combining the cognitive advantages of the simpler topology with the flexibility of the more complex one.

Rhetorical composites are specific constellations of (usually typed) nodes and links that form a logical unit for manipulation and navigation. For example, the Toulmin argumentation schema [TOUL69, STRE89] represents an argument as a composite of nodes that articulate a claim, its supporting datum, the warrant and backing that make the datum relevant to the claim, and any rebuttal. Derivatives of IBIS such as gIBIS focus on the basic tree consisting of an issue, various positions on that issue, and the arguments for and against each of the positions.[CONK87]

2.4. Element Standardization

The elements that we have discussed form the ontological foundation of hypermedia, suggesting that at least common terminology needs to be defined if standardization of any aspect of hypermedia is to be possible. This basic ontology is stable enough that the outlines of a reference model constructed now will probably be able to accommodate new techniques as they are developed, by adding subpoints as appropriate.

3. Implementation Concerns

Here we address both architectural and programming issues.

3.1. Layered Architecture

Architecturally, there is a growing consensus in favor of the value of a layered architecture for hypermedia. This approach has been applied both to data communications [DAY83] and the control of material handling [PARU88]. It not only permits modular, maintainable programs, but also facilitates access of a layered system by other systems that know the services published at each layer. Thus a layered architecture facilitates the development of hyperbases that can interact with one another as well as with users.

At least four layers are useful for a layered hypermedia architecture: data, element, inference, and interface.

3.1.1. Data

The data layer provides consistent data management for all information in the hyperbase, including both the contents of nodes and the links among nodes. If development and browsing of a hyperbase are to be separate processes, this layer manages access permissions to implement read-only networks. In a multiuser hyperbase, this layer must support multiple access with appropriate consistency management. Many applications will require it to support versioning as well. As hypermedia becomes more widely applied, distributed hyperbases will develop that will require the data layer to provide distributed data access, and in this case it would logically be defined as an RDA application on top of an OSI stack.

3.1.2. Element

The element layer provides separate services for managing nodes and links, and translates the raw data of the data layer into these atomic elements of hypermedia. The value of storing links separately from nodes is becoming evident, and is supported in Intermedia and in the link service furnished with Sun's Network Software Environment.[PEAR89] Among other benefits, this separation permits users to have private sets of links on a document, links that are not visible to other users. The link service needs to be able to combine different sets of links over a single document so that a user perceives them as forming a single set. Composites can be supported by appropriate internal recursion, thus permitting composites of any degree of nesting to be defined.

3.1.3. Inference

The inference layer provides at least the ability to traverse a link and retrieve the node at the destination. It is also a reasonable place to house services that do inference on source and destination nodes in conjunction with link traversal to support generalized link traversal as defined in [BIEB89].

3.1.4. Interface

The interface layer defines the mechanisms through which the user interacts with the hyperbase, and is responsible for displaying the information contained in the node.

3.2. Programming Issues

Object-oriented programming has been an important supporting technology for hypermedia, and the development of standards for OOPS will facilitate the interaction of various hypermedia systems.

Some systems, such as HyperTies [COGN89], HyperPAD [BRIG89], and HyperCard [WILL87], build nodes as a stack of different objects. A typical series of such objects includes the background, page, field, and button. If nodes are to be accessed through multiple systems, standardization of node architecture is necessary.

3.3. Implementation Standardization

Implementation standardization is necessary if hypermedia systems are to interoperate (for instance, by accessing the same information). A layered architecture offers promise as the reference model for such standardization. Outside of the hypermedia community, standardization in object-oriented languages and environments will greatly advance the foundation on which hypermedia systems rest.

4. Interface Issues

There are two main categories of interface issues in hypermedia: those concerned with constructing links among nodes, and those concerned with browsing a completed network. While many commercial systems include facilities for generating the contents of nodes, this process is so application-dependent that it seems to fall outside the scope of a reference model.

4.1. Building Links

Constructing the links is the most laborious part of populating a hyperbase. Three main sets of techniques are commonly used: automatic, mark-up and point-and-shoot.

4.1.1. Automatic Linking

Information retrieval (IR) techniques can be used to build networks automatically, for example, linking together all (textual) nodes containing a specified string of characters. Because these techniques are purely syntactical and do not "understand" the text, they must usually be supplemented by manual review and revision to eliminate spurious linkages and to add links that the syntactical scan misses. Natural language techniques from AI are beginning to improve the effectiveness of automatic linking, but still are not able to "understand" a text and so cannot completely eliminate manual editing.[HAYE88] Applied in real time, these techniques are a common way to implement virtual links. Standardization of IR techniques is marginally useful for the construction of links before run-time, since manual editing can correct any errors, but will be useful when these techniques implement virtual links, to insure consistent operation of such links across various implementations.

4.1.2. Mark-Up Linking

Many PC-based systems require manual mark-up with a text editor to identify link sources (and sometimes destinations). The most simple systems simply enclose link anchors in reserved brackets, which on execution are interpreted by the display manager and result in modified display attributes for the anchor. A more complex mark-up system, such as those conforming to [ISO86], provides a rich language for specifying functional components of a document, such as paragraph and chapter headers. While these mark-up languages are not originally designed for hypermedia, they provide a useful mechanism for facilitating automatic linking.

4.1.3. Point-And-Shoot Linking

The most sophisticated manual linking systems (for example, [PEAR89]) use a point-and-shoot interface that permits the user to point at the entities to become anchors and thus generate links directly.

4.2. Browsing

Browsing issues include the form and manipulation of the display, and navigational mechanisms.

4.2.1. Display

One area of active discussion in the hypermedia community is whether information should be divided into screen-sized chunks or "cards," or whether the screen should be treated as a window that moves over a larger unit of information. There appear to be applications where each approach is superior, and both should be accommodated in a reference model.

A number of issues concern the mechanics of manipulating the screen. For instance,

- In a scrolling system, does one push the window up over the information, or does one push the information up past the window?
- How does one select a link origin?
- How are active and inactive buttons represented on the screen?
- What is the correspondence between mouse action and cursor keys?

The Macintosh has provided a *de facto* standard for many of these issues. While standards are highly desirable (especially for users who must move from one platform to another), they are probably best handled in the broader CHI community, not by hypermedia specialists.

4.2.2. Navigational Mechanisms

Navigational mechanisms are of two main types: maps and path macros.

4.2.3. Maps

A map is a single display that shows nodes in abbreviated form (often as icons) and displays the links among them. While intuitive, a map can become cluttered and relatively useless for large, complex systems unless it is selective. For instance, a map displaying only links of a certain type and their associated nodes, or only composite nodes and not their components, will be simpler than a complete map.

4.2.4. Path Macros

A path macro is a composite that is generated in real time by gathering together nodes that the user has visited and the links along which they were visited, at least up to some limiting topology. For instance, a linear topology is commonly used to generate a backup stack. A path macro permits the user easily to revisit nodes that have been seen and are of particular interest.

4.3. Interface Standardization

Interface standardization is desirable, especially for people who must use more than one platform on a regular basis. Much of the desired standardization here will come not through work specifically in hypermedia, but through broader forums in CHI.

5. Conclusion

Hypermedia, especially in distributed applications, will benefit from standardization. To facilitate developing such standards, this paper has suggested a high-level reference model that describes the elements, implementation concerns, and interface issues for hypermedia. In the area of elements, the greatest need for standardization is in vocabulary. Implementation offers a rich possibility for standardization in the development of a layered model for hypermedia, and will profit from OOPS standardization being pursued elsewhere. Most of the interface standardization that is possible at this point is being pursued in the broader CHI community, and (apart from navigational devices that are particular to hypermedia) should not be the focal point of standardization efforts by the hypermedia community.

References

[BIEB89] M. Bieber and S.O. Kimbrough, "On Generalizing the Concept of Hypertext," Boston College Computer Science Department Technical Report BCCS-89-03, 1989.

[BIEM89] F.P.M. Biemans, "A Reference Model for Manufacturing Planning and Control," Ph.D. Dissertation, University of Twente, 1989.

[BRIG89] *HyperPAD User's Guide*, Brightbill-Roberts & Co., Ltd., Syracuse, NY, 1989.

[CATL89] T. Catlin, P. Bush, and N. Yankelovich, "InterNote: Extending a Hypermedia Framework to Support Annotative Collaboration," *Proceedings of Hypertext '89*, 365-378.

[COGN89] *Hyperties Author's Guide*, Cognetics Corporation, Princeton Jct., NJ, 1989.

[CONK87] J. Conklin and M.L. Begeman, "gIBIS: A Hypertext Tool for Team Design Deliberation," *Proceedings of Hypertext '87*, 247-252.

[DAY83] J. Day and H. Zimmermann, "The OSI Reference Model," *Proceedings of the IEEE*, 7 (December 1983), 1334-1340.

[HALA87] F.G. Halasz, "Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems," *Proceedings of Hypertext '87*, 345-366.

[HAYE88] P. Hayes, L.E. Knecht, and M.J. Cellio, "A News Story Categorization System," *Proceedings of the Association for Computational Linguistics Conference on Applied Natural Language Processing*, 1988.

[ISO86] *International Standard ISO 8879: Information processing -- Text and office systems -- Standard Generalized Markup Language (SGML)*, 1986.

[PARU87] H.V.D. Parunak and J.F. White, "A Synthesis of Factory Reference Models," *Proceedings of the IEEE Workshop on Languages for Automation*, Vienna (August 1987), 109-112.

[PARU88] H.V.D. Parunak and R. Judd, "LLAMA: A Layered Logical Architecture for Material Administration," *International Journal of Computer Integrated Manufacturing* 1:4 (1988), 222-233.

[PARU89] H.V.D. Parunak, "Hypermedia Topologies and User Navigation," *Proceedings of Hypertext '89*, 43-50.

[PEAR89] A. Pearl, "Sun's Link Service: A Protocol for Open Linking," *Proceedings of Hypertext '89*, 137-146.

[SMOL87] P. Smolensky, B. Bell, B. Fox, R. King, and C. Lewis, "Constraint-Based Hypertext for Argumentation," *Proceedings of Hypertext '87*, 215-246.

[STRE89] N.A. Streitz, J. Hannemann, and M. Thuring, "From Ideas and Arguments to Hyperdocuments: Travelling through Activity Spaces," *Proceedings of Hypertext '89*, 343-364.

[TOUL69] S.E. Toulmin, *The Uses of Argument*, Cambridge University Press, 1969.

[WILL87] G. Williams, "HyperCard," *Byte*, 12:14 (December 1987).

[WILL89] T.J. Williams, Editor, *A Reference Model for Computer Integrated Manufacturing (CIM)*, Instrument Society of America, 1989.

[ZELL89] P.T. Zellweger, "Scripted Documents: A Hypermedia Path Mechanism," *Proceedings of Hypertext '89*, 1-14.

An Interchange Format for Hypertext Systems: the Intermedia Model

Victor A. Riley

Institute for Research in Information and Scholarship (IRIS)
Box 1946
Brown University
Providence, Rhode Island 02912

ABSTRACT

Realization of the potential for information sharing that is inherent in hypertext systems depends on the ability to readily exchange data between those systems. A format for exchanging link-related data between first-order hypertext systems has been designed, and partially implemented, for the Intermedia system. The design is described to the individual field level. An example of usage for Intermedia link-related information is provided. The import, export, and verification utilities created for the interchange format are also described.

1. INTRODUCTION

The concept of hypertext has been around for several decades and recently we have seen the advent of several hypertext applications and systems. These applications allow one to create text, graphics, animation, video, and a number of other data types and proceed to link them together in any manner one sees fit. One capability that is still missing is the ability to transfer a set of hypertext links and documents from one system to another. Such a capability would open the door to sharing information and bring us one step closer to the mythical "hyperspace" or "docuverse" [Nels81] as Nelson has termed it. This paper examines a format for allowing interchange between hypertext systems.

2. PURPOSE OF THE INTERCHANGE FORMAT

Although a wide variety of hypertext/hypermedia systems exist today, they can be placed into one of two categories.

A *first-order* hypertext system manipulates the data of

- documents
- anchors within documents
- links between anchors
- some standard attributes associated with documents, anchors, and links. (The standard attributes include the name, creation time, and creator of a document, anchor, and link.)

Most hypertext systems in existence today are at least first-order hypertext systems [Conk87].

A *second-order* hypertext system manipulates all the information a first-order hypertext system contains with the additional support for

- user-defined objects and types
- user-defined attributes and keywords
- version history for documents, anchors, links, and attributes

There are only a few second-order hypertext systems in existence or development today: Engelbart's NLS/Augment [Enge68], Tektronix's Hypertext Abstract Machine [Camp88], and Nelson's Xanadu [Nels81].

Regardless of these categories, all hypertext systems need to store this persistent link data in some form of database. Since database formats and database files are inherently nonportable, a portable interchange format must be designed to facilitate exchanging sets of link-related hypertext data (what would be called *webs* in Intermedia).

Our interchange format contains the essential link-related information for a first-order hypertext system. Any application or system that understands the interchange format—what we call here a *participating* application or system—can capture all the existing hypertext link information as it exists in some other participating hypertext system. In conjunction with methods for converting and transferring document data, this capability makes possible the the complete sharing of information between hypertext systems, largely fulfilling the "docuverse" ideal.

The interchange format is useful for transferring data between similar first-order hypertext systems. It may also be useful for transferring first-order hypertext information into a second-order hypertext system or vice-versa. Suitable defaults could be supplied for the extra information necessary to transform first-order information into second-order; when transferring second-order information into first-order, the extra information could be ignored.

It needs to be stressed that the application-specific contents and format of hypertext documents themselves are outside the scope of the interchange format (which is concerned with the links between the documents) and of this paper. Data exchange on the document level is approached in other ways, commonly by adherence to a file format standard, such as PICT, TIFF, MacPaint, or RTF.

3. THE INTERCHANGE FORMAT

3.1 The Basic Objects

The information that most hypertext systems deal with is basically the same, although the names of objects may differ slightly from one system to the next. A first-order hypertext system deals with documents, anchors, links, and system attributes. These objects are stored in a database that the system's subordinate applications access in order to provide linking functionality. In the interchange format, each of these objects corresponds to a separate data file that contains the information specific to all occurrences of that object in the system. The architecture of these files is described in the next section.

Documents are the containers for the application-specific information in the hypertext system. They are built up of two components: the actual application-specific contents of the document (the information the user is interested in working with), and the information necessary for the application to render its views. The contents could be in the form of text, graphics, audio, video, etc.

Anchors are the locations in documents to which links are attached. Some examples of anchors are spans of text, graphical objects, audio or video, or bitmaps. Anchors are application-specific in that it is the application, not the hypertext system's database, that must render the anchor (e.g., in document views).

Links are the connections between anchors. They are directional in that they have a source and destination anchor. Applications can enforce bidirectionality or directionality by giving equal precedence to both source and destination, or keeping the distinction.

System attributes are predefined attributes that are associated with documents, anchors, and links. For all first-order hypertext systems, these consist of the name, creator, and creation time. Intermedia adds the modifier and last modification time to the standard system attributes.

User-defined attributes are also associated with documents, anchors, and links. They allow for flexible processing and retrieval of hypertext information.

3.2 Architecture of the Data Files

The interchange format consists of five data files for recording information about the link-related objects in the participating hypertext system, and one file for each document in the hypertext system.

document information *file*

The *document information* file contains general information dealing with all hypertext documents stored in the participating system. This information allows an application to gain access to the physical location of a document, get the user-defined access rights associated with the document, and retrieve information about the creator and last modifier of the document. A unique identifier for the document enables access to anchor information stored in the *anchor* file (described below).

anchor *file*

The *anchor* file contains information about all anchors in all documents in the hypertext system. This information allows an application to know where an anchor is located, who created and last modified the anchor, and other information that may be needed (e.g., to render a view of the anchor). A unique identifier for the anchor enables access to link information stored in the *link* file (described below).

link *file*

The *link* file contains information about all links between all anchors in the hypertext system. This information allows the system to traverse hypertext links. The file also contains information about the creator and last modifier of the link. A name and unique identifier for the link are provided, for consistency with the other files, and to allow for future expansion of functionality.

attribute definition *file*

The *attribute definition* file contains information defining the attributes and keywords used in the system. Predefined (system) attributes such as name, creator, modifier, creation time and modification time, are not defined in this file.

attribute *file*

The *attribute* file contains information about which objects have which attributes attached to them, as well as the values of those attributes.

document *files*

The format of each document file is determined by its contents, and the requirements of the participating application in which it is used. Formats currently employed in Intermedia include "web," formatted text, structured graphics, timeline, and bitmap image. As noted above, the exchange of this information between systems is not intended to be part of the interchange format. However, several fields in the five link-related files are indirectly dependent on the existence, system attributes, or contents of the document files. These are described under "Implementation."

3.3 Implementation

This section describes the interchange format at the level of data formatting and field definition. Examples illustrating these descriptions are provided in Section 4.

Data Formatting

In order to make the interchange process as straightforward as possible, the format of the data to be exchanged is kept simple

Each value is stored in normal ASCII format, so that it is easily readable, editable, and portable.

Each data record in a file is delimited by a carriage-return/linefeed character pair. Each data field in a record is delimited by a tab character. To avoid conflicts, the tab character is not permitted in document and path names.

Data values are either strings or numbers. String values can be any length. Numeric values are four full bytes; the decimal ASCII digits correspond to an unsigned 32-bit long word. Certain numeric fields store information in terms of the bit patterns in the long word.

All numeric values that denote a time are stored in Unix GMT format, which expresses a time value as the number of "ticks" since an established starting point (midnight of January 1, 1970). There are about 31.5 million ticks in a calendar year.

Values for the predefined system attributes (*creationTime, modTime, creator, modifier,* and *name*) are obtained from the operating system via the Export utility.

Since some applications may require data not specifically identified in the interchange format, certain fields are allotted for this special purpose. Data in these fields is arbitrarily stored in string format, for maximum flexibility, and may need to be converted to some other data format for use by a target application. This feature allows for a variable number of data values and types to be transferred by the interchange format.

Site Identification

The first field of each record contains a site-specific ID. This value is composed of a unique number for each site (or machine) using the interchange format and a site-unique number for the database to which hypertext data is being imported or exported. The combination of a siteID (with its "site" and "database" components) and an object's own unique ID allows the object to permanently maintain its identity across exchanges of data between sites.

Some type of assignment of unique numbers for sites must be administered in order to implement this feature fully. If this were not done, however, the remainder of the interchange format could still be implemented independently.

Another uniqueness scheme might consist of combining a 32-bit random number with two 16-bit random numbers, which would provide IDs for the site and the local database, respectively. This 64-bit number should be unique across the domain of all hypertext systems.

Field Definitions

document information *file fields*

siteID	(Numeric) Unique identifier of the originating site and database.[1]
docID	(Numeric) Unique identifier of a document. Assigned sequentially by the DBMS.
docType	(Numeric) Code specifying the document's type. Allows the system to identify the the correct target application for application-specific data.[2]

---

[1] The first short word of the value stores the site number; the second short word stores the database number. The interchange format stores the number resulting from reading the two short words as a long word.

[2] Intermedia supplies codes for its currently supported document types (InterWord, InterDraw, etc.). Codes must be standardized for participating systems, be these numeric codes or string codes.

accessRights	(Numeric) Number expressing the types of access allowed to the document for various groups of users.[1]
groupName	(String) The name of the group identified in *accessRights*.
creationTime	(Numeric) Time the document was created.
modTime	(Numeric) Time the document was modified.
creator	(String) Name of the user that created the document.
modifier	(String) Name of the last user to modify the document.
docName	(String) Name of the document. Assigned by user when document is saved.
path	(String) Directory location of the document in the Unix tree, relative to the application's home directory.

***anchor* file fields**

siteID	(See description for *document information* file.)
anchorID	(Numeric) Unique identifier of an anchor; assigned sequentially by the DBMS.
anchorDocID	(Numeric) Value of *docID*, in the *document information* file, for the document containing the anchor identified by *anchorID*. Allows system to determine the document in which the anchor is located.
creationTime	(Numeric) Time the anchor was created.
modTime	(Numeric) Time the anchor was modified.
creator	(String) Name of the user that created the anchor.
modifier	(String) Name of the last user to modify the anchor.
anchorName	(String) Name of the anchor.
X-loc	(Numeric) X, Y, and Z-axis coordinates of the anchor, within the document specified by *docID*.
Y-loc	These allow system to determine placement of anchor in document window.
Z-loc	Interpretation of coordinates is application-specific.
appData	(String) Application-specific information dealing with anchors. Allows participating application to obtain other information required. Examples might include data needed to render a type of window view. Values are separated by space characters, or other delimiters specified by the participating application.

***link* file fields**

siteID	(See description for *document information* file.)
linkID	(Numeric) Unique identifier of a link; assigned sequentially by the DBMS.
linkType	(Numeric) Code specifying the type of relationship between the link's two anchors.[2]
srcAnchorID	(Numeric) Source anchor of the link, as identified by the value of *anchorID*, in the *anchor* file.

---

[1] The four bytes of the value, from high to low, correspond to the rights granted to: system administrator, owner, group, and world (all) users. The bits of each byte, from high to low, correspond to the following rights granted to each of the four user groups: change access rights for the document, write to the document, create links in the document, and view the document. The bits are set on or off in groups of two.

[2] Intermedia supplies codes for its currently supported document link types. Codes must be standardized for participating systems, be these numeric codes or string codes.

destAnchorID	(Numeric) Destination anchor of the link, as identified by the value of *anchorID*, in the *anchor* file.
creationTime	(Numeric) Time the link was created.
modTime	(Numeric) Time the link was modified.
creator	(String) Name of the user that created the link.
modifier	(String) Name of the last user to modify the link.
linkName	(String) Name of the link.

***attribute definition* file fields**

siteID	(See description for *document information* file.)
attDefID	(Numeric) Unique identifier of an attribute definition; assigned sequentially by the DBMS.
attDefType	(Numeric) Code specifying the attribute's type. [1] General-purpose flag value. One potential use is to specify what objects the attribute can be attached to.
attName	(String) Name of the attribute.

***attribute* file fields**

siteID	(See description for *document information* file.)
attDefID	(Numeric) Value of *attDefID*, in the *attribute definition* file. Allows system to look up the attribute's name and type.
attValType	(Numeric) Code specifying the data format of *attValue*.[2]
attValue	(Variable format) Value of the attribute. Assigned by the user. The next three fields refer to the object to which the attribute is attached: (document, anchor, or link).
objectType	(Numeric) Code specifying the object type (document, anchor, or link). [3]
objSiteID	(Numeric) Value of *siteID*, in the corresponding file (*document information*, *anchor*, or *link*).
objectID	(Numeric) Value of the object's *ID*, in the corresponding file (*document information*, *anchor*, or *link*).

4. EXAMPLE OF USE

This section illustrates how the interchange format can be used to create, store, and reuse link-related data from a first-order hypertext system, namely Intermedia.

4.1. Sample Data In Intermedia

The Intermedia system is described in a number of articles, notably [Meyr86] and [Yank88a]. A public release of the software, with full documentation, is also currently available through IRIS and through the Apple Programmer and Developer's Association (APDA). This release (3.0) runs on the Apple Macintosh II, under version 1.1 of A/UX, Apple's version of Unix.

Figure 1 shows the Intermedia desktop environment. Two elementary sample documents have been created, one in Intermedia's InterWord format, the other in InterDraw. For the clarity of the example, these objects have been created in an empty new Intermedia database. The folder window (labelled "/int/docs/demo") contains the icons representing the documents and the Web comprising the links between them. The Web View window displays the linking structure. The information used in generating this Web View is also used to generate the *anchor* and *link* files of the interchange format.

---

[1] A participating system supplies codes for its currently supported attribute definition types. Codes must be standardized for participating systems, be these numeric codes or string codes.

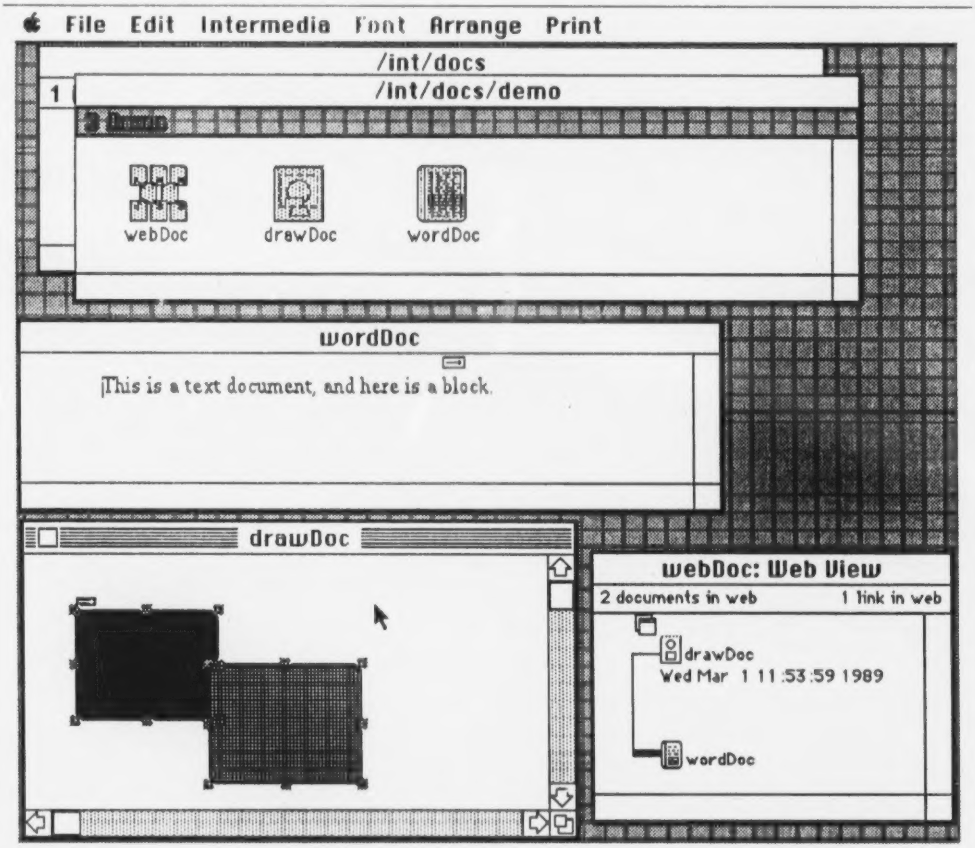
[2] A participating system supplies codes for its currently supported attribute value types. Codes must be standardized for participating systems, be these numeric codes or string codes.

[3] A participating system supplies codes for the object types of document, anchor, and link. Codes must be standardized for participating systems, be these numeric codes or string codes.

An anchor has been created from the word "block" in the InterWord document (indicated by the arrow marker over the word). Another anchor has been created from the two rectangles in the InterDraw document. Each anchor can be assigned a name; the names are not shown here, but can be viewed and edited by the user by means of dialog boxes.

The current version of Intermedia does not make use of attributes and keywords, so these are not represented in the example.

At the moment shown in the figure, the link between the two anchors has just been followed, from the InterWord to the InterDraw document. This is shown by the shaded selection handles around the rectangles and the shaded link line in the Web View.



This is a text document, and here is a block.

2 documents in web

1 link in web

Wed Mar 1 11:53:59 1989

Figure1. Sample documents on the Intermedia desktop. Linking is indicated by the arrow markers in the document windows and the icon-connecting line in the Web View window.

Intermedia allows users to edit the access rights to documents, through the use of the "Document Properties" dialog box (simple matrix of sixteen check boxes, not shown here). The ability to edit these rights is itself controlled by the rights scheme, with the system administrator having ultimate control over a document's access. The rights for the two documents in this example are set so as to grant the system administrator, document owner, and members of the owner's "group" the right to perform all operations on these documents; all other users (the "world") can only read them and make links in them.

4.2. Sample Data in the Interchange Format

This section illustrates how a current version of the interchange format stores the first-order hypertext link information embodied in the sample Intermedia environment in Figure 1.

After creation of the documents, anchors, and links in Intermedia, the link-related information stored in the Intermedia database is converted into the interchange format by use of the Export utility, which is described in Section 5.

The ASCII data values resulting from this conversion are shown in the following tables, as they would appear when viewed in a text editor (minus their field and record delimiter characters). These values fully describe the anchor, link, and document-properties information contained in the Intermedia database for the documents depicted in Figure 1.

It is arbitrarily assumed that the ID numbers for both the current site and converted database are 1. Using the rule for generating the value of the *SiteID* field noted under "Implementation," the following long word is stored:

00000000 00000001 00000000 00000001
site number database number

This is displayed as the number 65537. Note that this value is the same for every data record in the example.

document info *file*

Field	**Value**	**Value**
siteId	65537	65537
docID	1	2
docType	300	301
accessRights	4294967055	4294967055
groupName	iris	iris
creationTime	604771573	604771642
modTime	604771573	604771642
creator	var	var
modifier	var	var
docName	wordDoc	drawDoc
path	demo	demo

The documents in the example were the first created in the Intermedia database, so their *docID* numbers are "1" and "2".

The *docType* uses Intermedia type codes: "300" for InterWord, "301" for InterDraw.

The *accessRights* are stored in the bit pattern of the value's long word. The value for the documents in this is written in ASCII as "4294967055," which is equivalent to the bits:

11111111	11111111	11111111	00001111
system	owner	group	world

Using the rule noted under "Implementation," system administrator, owner, and "group" users can perform all operations on these documents; "world" users can only read them and make links in them.

The *groupName* of the group referred to in the *accessRights* is "iris". The *creator* and *modifier* fields contain the user ID of the author of this example: "var".

The *creationTime,* expressed in Unix GMT format as "604771573," is Wednesday, March 1, 1989, 4:06:13 PM.

The *docName* values of the two documents are those shown in the documents' windows in Figure 1. The relative *path* name of the document files is that shown in the folder window in Figure 1.

anchor *file*

Field	**Value**	**Value**
siteID	65537	65537
anchorID	1	2
anchorDocID	1	2
creationTime	604771726	604771729
modTime	604771726	604771729
creator	var	var
modifier	var	var
anchorName	Source Anchor	Destination Anchor
X-loc	40	23
Y-loc	45	28
Z-loc	0	0
appData	1	1 2 0 3

The anchors in the example were the first created in the Intermedia database, so their *anchorID* numbers are "1" and "2". Their *anchorDocID* values identify the documents they were created in: "1" (the InterWord document) and "2" (the InterDraw document), respectively.

The *anchorNames* of the anchors are "Source Anchor" and "Destination Anchor". These names are informational; they do not affect the directionality of the link.

The X, Y, and Z coordinates for each anchor, and the values in the *appData* field, are interpreted by the applications associated with the documents identified in the *anchorDocID* field (InterWord and InterDraw), in ways dependent upon the document contents. For instance, the data value for anchor 1 specifies the "anchor type," while the values for anchor 2 specify: the two objects the anchor is connected to, the "view index" of the anchor, and the "mark type" (these terms are included for illustration; their definition is outside the scope of this paper). Other link-related data values that do not fit elsewhere in the architecture of the interchange format can be recorded here in similar fashion.

link file

Field	**Value**
siteID	65537
linkID	1
linkType	2
srcAnchorID	1
destAnchorID	2
creationTime	604771731
modTime	604771731
creator	var
modifier	var
linkName	Demo Link

The link between the anchors in the two documents in the example was the first created in the Intermedia database, so its *linkID* number is "1".

The *linkType* uses Intermedia type codes: "2" denotes a "reference" link.

The "source" anchor of the link is the one identified in the *anchor* file by the *anchorID* of "1"; consequently "1" is stored here for *srcAnchorID*. The "destination" anchor of the link is treated in parallel fashion. Keep in mind that linking in Intermedia is bidirectional; the distinction between source and destination is maintained for participating systems that distinguish between the two.

The *linkName* of the link is "Demo Link". This value is not presently used in Intermedia, but is stored for consistency, in the event it is needed for a future version of Intermedia, or for another participating system.

There are a number of other fields in the interchange format that are used this way, providing flexibility beyond the bare needs of Intermedia itself. *SiteID*, and the *creationTime, modTime, creator,* and *modifier* fields in the *anchor* and *link* files are examples.

attribute definition and attribute files

Although attributes were not included in this Intermedia example, their use in this context can be illustrated hypothetically.

For instance, in order to support optional unidirectional linking, an attribute with the *attName* of "anchorType" could be entered in the *attribute definition* file. Codes for "source" and "destination" could then be entered as values for *attValue* in the *attribute* file, and attached to particular anchors by making the requisite entries for *objectType* and *objectID*.

Another significant use of user-defined attributes is for filtering of hypertext information based on *keywords*, which are text strings attached by the user to hypertext objects. Keywords serve as flags for associating objects with each other. Typical keywords might be "Modernism," "Mitosis," "Moon," or "Manichean." Keywords can be implemented by defining an attribute named "Keyword" and allowing users to enter their keywords as values for the attribute.

document files

The operating system files that store the contents of the Intermedia documents shown in Figure 1 are located in the directory identified in the *path* field of the interchange format's *document information* file. The names of the document files are stored in the *docName* field of the same interchange format file.

As noted in Section 2, the application-specific contents and format of the document files are not considered part of the interchange format. In order to support such exchange of document information, Intermedia provides various methods for importing and exporting document content data. These methods include the use of standard file formats, such as RTF (for InterWord documents), PICT (for InterDraw documents), and TIFF or MacPaint (for InterPix bitmap images).

4.3. Other Intermedia Usage of the Interchange Format

An early version of the interchange format has already been used in the suite of "Webware" products making up part of the public release version of Intermedia. The procedure for installing the webs for "Exploring the Moon" and the Intermedia Tutorials into the Intermedia link server database involves running a script that calls the Import utility, which transfers web data in the interchange format from a floppy disk to the Intermedia server hard disk. The Import utility is described in Section 5 of this paper.

This early prototype of the interchange format does not support attributes or SiteIDs, and the storage for anchors is tailored to their treatment within Intermedia.

5. UTILITIES FOR THE INTERCHANGE FORMAT

A number of utilities have been created for use with the interchange format. Some of the utilities process the data of the interchange format to validate the data, others are used in conjunction with the the Intermedia Link Protocol Server ("the link server") to import and export data into the Intermedia database.

5.1. Verify

The Verify utility checks the consistency of the interchange format files. It ensures that all documents exist for all anchors, and that all anchors exist for all links. If keywords are implemented, the utility ensures that all documents, anchors, and links exist for all keywords. A series of hash tables is used during the checking process. If any ID is not in the hash table, the object being processed is removed and placed in an error file, and the user is informed.

5.2. Export and Import

The Export and Import utilities are used to extract and store, respectively, the data from Intermedia's database using the link server.

Earlier prototypes of these two utilities were helpful in the conversion of our Intermedia databases when we exchanged Ingres for the Intermedia link server and its new database system based on C-Tree [Fair88]. The utilities have also helped us convert databases from one data dictionary format to another, by running Export with an old-format server, and Import with a new-format server.

The Import utility reads the files of the interchange format and calls the import functions of the link server to add the data to the database. One parameter to the utility specifies whether to create new IDs for each object being added to the database or to reuse the existing object IDs. This feature allows us to either append data to the end of the database (with new IDs), or replace the data in the database with new data (having the IDs of the existing objects). Using the "replace" feature we are able to change the location of the document tree without having to change the IDs for the documents. The other parameters to the utility specify the Unix file system locations for the location to read the interchange format from, the name of the database to add the data to, and the new location for the document tree.

The Export utility calls the export functions of the link server to dump all data from the database into the interchange format. The Verify utility can be run in conjunction with Export, to ensure data integrity. The parameters of Export are the same as those of Import that deal with Unix file specifications, except that Export writes where Import reads, and vice versa.

5.3. Future Developments for Utilities

The utilities described here have been integrated into an application that will potentially be in a publicly available version of Intermedia. This application, called Transfer, enables users to select document, anchor, and link information to be exported by selecting folders and their contents (i.e., documents and webs). In order to maintain the integrity of all the webs in the selection, documents that lie outside the selection in the folder hierarchy, but have links or anchors in a selected web, are also exported. When exporting, the user can select the type of media to export the data to. Hard disk, floppy diskette, and tape are currently supported. Users can also import previously exported data, from the same media types.

At present, the Transfer application generates data in a form of the interchange format described here. It is intended that the application be able to generate any of a number of other formats as their definition and use becomes available.

There are also plans to create other utilities to enable the conversion of first-order interchange formats into second-order interchange formats, or from prototype first-order interchange formats into production first-order interchange formats, as their needs arise.

6. OTHER INTERCHANGE FORMATS

At the time I developed the interchange format described here, I knew of no other hypertext interchange formats under development. Many design elements in this interchange format apply specifically to the requirements of the Intermedia system.

However, the major conceptual elements are common to most other hypertext interchange formats.

In this described interchange format, the structure of the data file is static, while the the data that fills that structure changes dynamically. A format like this is very simple to implement. However, when interchanging with other disparate systems this interchange format becomes very difficult to use. Converting its structure to a tagged format, like SGML, would make it more portable.

It should be possible to convert this format to the X3V1.8M interchange format [Gold89] with relatively few or no extensions to the HyTime DTD. However, there are several drawbacks in doing this. First, none of the documents in Intermedia are stored in SGML format, so references to components of the documents may be difficult. Second, the link-and-anchor database is separate from the document database, in order to support linking to non-writable media (like CD-ROM disks) and to support multiple web mappings over the same document sets.

The task of converting this data structure to support any of the interchange formats [Born89] that conform to the Dexter model [Hala89] would be possible as well. This would require adding tags and attributes the the existing data elements with some minor reorganizations. This is planned as a future project.

6. SUMMARY

In this paper a format is documented that shows the structure of the data files and the minimum information necessary to transfer hypertext information from one first-order hypertext system to another. These data files, when combined with a methodology for converting and transferring the contents of application document files, embody an interchange format enabling the full exchange of information between existing hypertext systems. This was demonstrated by the use of the interchange format to transfer data into and out of Intermedia.

It is hoped that this format could be a base of ideas in developing an interchange standard for first-order hypertext systems thus enabling the sharing of hypertext information more freely.

The need remains to establish and publish conventions for assigning values in the *SiteID, docType, linkType, attDefType, attValType,* and *objectType* fields, to insure compatibility between the systems on both ends of a data exchange.

8. ACKNOWLEDGEMENTS

I wish to thank everyone at IRIS for their help during the writing of this paper, especially Jim Coombs and Norm Meyrowitz for being the best sounding boards for my ideas, and Mark Sawtelle for assistance in preparing the text.

REFERENCES

[Born89] J. Bornstein, "Hypertext Interchange Format—Discussion and Format Specification—DRAFT 1.3.3", September 1989. Available from author.

[Camp88] B. Campbell, J. Goodman, "HAM: A General Purpose Hypertext Abstract Machine," *Communications of the ACM*, **31**(7):856-861, 1988.

[Conk87] J. Conklin, "Hypertext: An Introduction and Survey," *IEEE Computer*, **20**(9):17-41, 1987.

[Enge68] D. Englebart, W. English, "A Research Center for Augmenting Human Intellect," *Proceedings of FJCC*, **33**(1):395-410, AFIPS Press, Montvale, NY, 1968.

[Fair88] FairCom, *c-tree™ File Handler Programmer's Reference Guide*, FairCom, Columbia, MO, May, 1988.

[Gold89] C. Goldfarb, A. Talbot, *Journal of Development, Part Two: Standard Music Description Language (SMDL) Hypermedia/Time-based Document Subset (HyTime)*. ANSI X3V1.8M, The Computer Music Association, P.O. Box 1634, San Francisco, CA 94101-1634, October 31, 1989.

[Hala89] F. Halasz, M. Schwartz, "The Dexter Hypertext Reference Model", to be presented at the NIST Hypertext Standardization Workshop on January 16, 1989.

[Meyr86] N. Meyrowitz, "Intermedia: The Architecture and Construction of an Object-Oriented Hypermedia System and Applications Framework," *OOPSLA '86 Proceedings*, **21**(11):186-213, ACM SIGPLAN Notices, November, 1986.

[Nels81] T. Nelson, *Literary Machines: The Report on, and of, Project Xanadu, Concerning Word Processing, Electronic Publishing, Hypertext, Thinkertoys, Tomorrow's Intellectual Revolution, and Certain other Topics Including Knowledge, Education, and Freedom*, Swarthmore, PA, 1981. Available from author.

[Yank88a] Yankelovich, N., Haan, B., Meyrowitz, N. and Drucker, S., "Intermedia: The Concept and Construction of a Seamless Information Environment, *IEEE Computer*, **21**(1):81-96, 1988.

Strawman Reference Model for Hypermedia Systems

Craig W. Thompson

Information Technologies Laboratory
Texas Instruments Incorporated
P.O Box 655474, MS 238
Dallas, Texas 75265
Email: thompson@csc.ti.com Telephone: (214) 995-0347

Abstract

This paper provides a strawman reference model that can be used for comparing and reasoning about hypertext/hypermedia systems. It begins with a glossary of hypermedia terms. Agreeing on these provides a common vocabulary for developing the reference model. The reference model itself is presented in terms of *basic features* all hypermedia systems have, *advanced features* some hypermedia systems have, and *open features* that hypermedia systems share with other computer systems. These features represent independent dimensions which can be used to classify or compare existing hypermedia systems and to contrast them with near-miss related systems. Based on the features, the architecture of an ideal hypermedia system is described that covers existing hypermedia systems. The architecture is modular. A consequence is that discussion of standards or a more detailed reference model can focus on one module at a time, avoiding movement toward a portmanteau standard. The final section of the paper evaluates some areas where consensus and eventual standardization of hypermedia systems is possible and would be valuable. An appendix references some standards related to hypermedia systems. Another appendix is an initial document log listing references important to hypermedia standardization.

BEST COPY AVAILABLE

1 Introduction

The premise of the Hypertext Standardization Workshop is that "hypertext and hypermedia technologies have reached the point where it makes sense to consider their potential for formal standardization" [Workshop Call for Papers].

This paper provides a strawman reference model that can be used for comparing and reasoning about hypertext/hypermedia systems and suggests some areas where enough consensus could occur to make eventual standardization possible.

Section 2 provides an (incomplete) glossary of hypermedia terms. A *standard glossary* would provide a common vocabulary for implementors and users of hypermedia systems. This level of standard promotes communication among *people*.

Section 3 presents a strawman *hypermedia reference model.* Standardizing on a reference model should make it possible for *people* to compare different hypermedia systems and other closely related systems. The section demonstrates this by using the dimensions of a hypermedia system described in the reference model to compare several hypermedia systems. The section concludes with an ideal, modular architecture for a hypermedia system.

Operational standards should make it possible for *computer systems* to share data or interface to each other. Section 4 evaluates potential areas, indexed to the reference model, where operational standards for hypermedia systems may be possible and would be valuable.

Appendix A references some existing standards related to hypermedia systems. Appendix B is a place holder for the document log that a hypermedia systems study group would maintain.

In fact, overall, this paper can be viewed as the skeleton for a Final Report of a study group yet to be formed recommending whether and what hypermedia standardization is useful. Such a report might lead to the formation of an official standards body charged with formulating detailed hypermedia standards.

2 Glossary

The purpose of the glossary is to register terms and how they are used in different hypertext systems. The value of a glossary in standardization is to provide a common vocabulary so we all understand common terms the same way and can distinguish their various overloaded meanings. In addition, glossary terms are important in the development of a reference model (section 3) and provide a simple approximate way to scope a domain. Here we only list some of the more prominent terms that need to be defined.

`hypertext`
`hypermedia`
`browser`
`editor`
`hypermedia abstract machine`
`unique id`
`node`
`cut-and-paste`
`link`
`warm link`
`hot link`
`field`
`button`
`anchor`
`link service`
`link protocol`
`content`
`annotation`
`version`
`configuration`
`web`
`network`
`guideline`
`stack`

216

`card`

`background card`

`field`

`locktext`

`script`

`scroll`

`bookmark`

`history`

`map`

`open architecure`

Here we only comment that some terms like *link* are heavily overloaded. Other terms like *node*, *card*, *frame* are system-specific names for the approximately the same concept.

3 Reference Model

A hypermedia *reference model* is an English description of characteristics that "cover" existing (and future) hypermedia systems and provide *people* with a way to compare them.

Subsections 3.1, 3.2, and 3.3 sketch basic, advanced, and open features of a prototypical hypermedia system. Each feature represents an independent dimension in which hypermedia systems vary. Subsection 3.4 compares how some existing hypermedia systems fit this model and how some near-miss systems compare. Subsections 3.5 and 3.6 describe an "ideal" architecture for a hypermedia system based on the premise that *orthogonality implies modularity*. If this premise is correct, we should expect to concentrate standardization efforts on modules, not on whole systems.

3.1 Basic Features

All hypermedia systems have the following basic characteristics or dimensions through which they vary and can be compared.

The *representation dimension* provides the primitive *media types* or content part, and the compositional *data model* or structural part, that together are used to represent information in a hypermedia system. It is convenient to distinguish these two sorts of representations as separate dimensions.

Media Types. A *hypertext* system must be able to represent text (as well as structure). A *hypermedia* system adds other media types (bitmaps, graphics, sound, video). Specialized media editors are needed to permit WYSIWYG editing of media types. Compression of media types may be supported; automatic conversion between some media types is supported (e.g. graphic-to-bitmap). (Various) standards already exist for representing many of these media types (see Appendix A).

Data Model. A *data model* provides the structuring primitives[1] of the hypermedia system. Together, the data model and media data types are used to *represent* or encode the application-specific information content in a hypermedia information system. Specialized hypermedia interpreters, usually with built-in operations, operate on the basic data structures of the data model.

Data modeling is *the* most interesting and diverse dimension of hypermedia systems. *The* common invariant that all hypermedia systems share is the notion of navigating through an information space by following *links*. Beyond that, systems vary widely, most implementing some sort of semantic net with more or less structure. Many hypermedia glossary terms describe system-specific data model concepts (e.g., stack, card, history). *Nodes* may be inherently unstructured; they may have built-in or user programmable types; or they may have attributes, fields, or buttons. Links also vary. Most are binary; they may be typed and have attributes; they may anchor at nodes or within nodes in a media- or type-specific or application-specific way; or they may be built from lower level primitives (*anchors* and *go to*'s as in Hypercard).

While data models differ across different hypermedia systems, they are nearly always built-in to today's systems. Later, in section 4 we will consider when and whether mappings between different data models are possible.

User Interface. The *user interface* provides the capability of viewing and editing (WYSIWYG) presentations of information represented by the data model and media types.

---

[1] the *hyper* in hypermedia

Some hypermedia systems like KMS and HyperCard use the metaphor of a "notecard" and only provide fixed (screen-sized) cards with only one card visible at a time. Others like NoteCards use an overlapping or tiled window system metaphor of flexible-sized cards with the content and structure of a card still tied one-to-one to the display window. Guide provides *scrolling* and *progressive disclosure*, a step towards providing the user with control of which objects he can see on a screen. More generally, a many-many view mapping like that in CMU Andrew covers all of the above cases.

Persistence. Hypermedia systems all provide some notion of transferring application-specific content and structure to and from some *persistent storage repository.* They vary on the *unit of transfer* (e.g. Guide document, HyperCard card, Notecards application) and the file or database format they use to encode the data represented by the data model.

3.2 Advanced Features.

Not all hypermedia systems have the following advanced characteristics. While not mandatory (essential, intrinsic, defining), they complement the basic features and are needed for non-trivial hypermedia systems.

Multi-user. Computer-supported cooperative work requires many users to access shared data. Some hypermedia systems support this. Sharing by multiple users adds the need for some *concurrency control scheme* like *locking* or *time-stamping* so users can coordinate access to shared data. Data and/or structure may be *read-only* or *modifiable* according the *access rights* of users. Users can be *granted* different access rights at different times or for different purposes.

Distributed. Even for a single user, hypermedia data may be stored in a central repository or be distributed. For instance, content may be on a WORM device and structure may be stored in a relational database.

Uniform Representation[2] Many hypermedia systems make a distinction between node and contents. This forces the user to "chunk" the information he wants to represent into some fixed grain-size. This can lead to users spending time manually restructuring information. Advanced systems provide a more recursive formulation of the data model allowing content to contain nodes

---

[2] This feature is not independent of the data modeling feature presented earlier but is included here as a major dimension for comparing advanced hypermedia systems

(further structured information). This extra information plus a richer mapping of the more uniform data model to the user interface can give the user many *views* of the same information. Systems like Guide begin to take advantage of this by allowing the user to control which objects are visible using *progressive disclosure*. Intermedia *webs* allow two or more views to "share" common nodes. Systems like Lotus Agenda allow the user to reorganize the information based on a simple form of computed view. The semantics of sharing common objects from different perspectives can lead to dangling pointers and view update problems.

A different aspect of uniform representation involves the ability to deal with *foreign* nodes. These are nodes whose contents are opaque to the hypermedia system. For at least two reasons, uniform representations must generalize to account for these foreign representations. First, not all workstations can display all information, so video or even graphic information will remain opaque on these workstations. Second, hypereditors like KMS or Neptune can bind to non-hypereditors, like word processors, that do not understand link protocols (are not themselves uniform; do not represent their internal information in a way the hypermedia system can interpret). In this case, links typically anchor to whole nodes, which act to "wrap" the foreign editor, or else link anchors consist of two parts, a node id and a *specifier*, often written in a script language that can be interpreted by a foreign tool, telling how to offset into the foreign representation. Sun provides an application-independent Link Service protocol for standardizing cross-application linking as does HP New Wave.

One last variation in representation is whether hypermedia systems permit users to define the scope of objects like figure, section, document, library, video clip, or whether these types are built-in.

Computational Completeness. The computational completeness dimension describes how procedural information can be associated with the hypermedia data model to model behavioral aspects of the information.

Procedures can be coupled with data in many ways. Most characteristically, an anchor contains a script (procedure in a language specialized to the data model as in HyperCard) that is triggered if the anchor is activated. Alternatives are demons and rules as in Object Lens, procedures in general purpose languages as in NoteCards, assertions, and so on. Since procedural attachments are added dynamically, there must be an interpreter or dynamic compiler.

This dimension is *the* hardest to transport across systems, as we discuss in section 4.

Query. Hypermedia information spaces are often large. *Navigation* is used for browsing; *bookmarks* for going to known places. *Search* is used for locating items of interest by their characteristics. Some dimensions of search include limiting the scope or order of a search; string search versus indexing text; boolean search predicates and their possible use of indices; user-defined search predicates; incremental search; and how the end user can easily specify complex searches.

Another dimension involves what to do when search is successful. Alternatives are that the search results in a computed *path* through the information space or in a new view of the information space. Much work from the database and information retrieval areas is useful here. Query is a very rich dimension.

General-purpose procedural attachments generalize query capabilities and many hypermedia systems contain weak or no specialized query facilities. This leaves the burden of specifying complex queries to the user via programming.

Versions, Configurations. Especially for design applications (e.g., documents, software), where the life cycle of a design needs to be represented, a *Change Management* data model consisting of *versions, configurations* and *transformations* is useful for recording change, how a complex object is composed of its parts, and how change propagates.

Portmanteau Features. Subsection 3.4.2 describes near-miss systems closely related to hypermedia systems. We can mine these systems for other characteristics that hypermedia systems could have.[3] This could overload hypermedia systems with more than their ordinary meaning but the exercise is needed to determine how these systems differ from hypermedia systems.

3.3 Open Features

Open features are generic and belong to many or all computer systems. They may apply in special ways to hypermedia systems.

---

[3] For example, few if any hypermedia systems provide *parsers* to automatically recognize structure in unstructured information. This is clearly important since a whole hypermedia business could be built around structuring the mass of unstructured information. Most parser technology is aimed at recognizing already designed languages. The Oxford English Dictionary project at University of Waterloo is one place to look for good ideas on the interplay between parsing, querying, and computed data models induced by a grammar.

Human Factors. This dimension measures how likeable, usable, and effective a system is for the tasks it is designed or needed for.

Open versus Closed Architectures. Hypermedia systems vary along the dimension of how closed or open they are; that is, how extensible they are. Some aspects of openness are:

- none – browsers
- editing only – simple authoring systems like Guide
- user can add node and link types; or can specialize classes the system defines.
- user can provide procedural attachments
- system has an application program interface
- system is modular and modules can be replaced

Monolithic versus Modular Architecture Today's hypermedia systems are monolithic. An alternative is a modular, toolkit architecture in which modules can be added or replaced as the need arises. This would mean that design applications could make use of the change management module but other applications would not have to pay this cost. If some specialized change management is needed, only that module is replaced. The modules themselves may be open–e.g., the query optimizer could be programmable; the version scheme's notion of deltas could be too; pragmas might control how objects are clustering on disk; new kinds of presentations could be added to the user interface. A key issue related to modularity involves determining the protocols an existing foreign editor must implement to become a friendly hypereditor. It is more likely that "the world's best editors" can be modified to be hypermedia-conformant than that hypermedia editors will come to rival these editors.

Portability and Industrial Strength. The portability dimension describes how a system is bound to its environment and how easily it can be moved to other environments. It will be more portable if 1) it is implemented on de facto standard, industrial strength platorms (Unix, DOS, X-Windows, C++, SQL, etc), 2) it contains alternative, equivalent implementations for different environments (Open Look versus Presentation Manager), and 3) it can exchange data with many existing, popular data exchange formats.

A hypermedia system is industrial strength if 1) it is debugged and maintained, 2) it scales up for large hypermedia bases, 3) performance is acceptable, 4) it has (online) documentation and tutorials, 5) it is portable, and/or 6) it is being used in practice.

Cost, Availability, Service. The world's best designed hypermedia system is worth less if it is too costly, unavailable, breaks, and so on. This dimension is a non-technical road block to many systems.

Packaging. This characteristic represents the particular binding of all previous characteristics that defines any given system. It is measured by some sort of *success* metric.

3.4 Comparison of Existing Systems

If the reference model just defined is successful, we should be able to compare existing and related systems using the dimensions it defines.

3.4.1 Comparison with Other Hypermedia Systems

Figure 1 makes this comparison for existing hypermedia systems.

	HyperCard	Notecards	Guide/IDEX	Intermedia	KMS
media types	bitmaps text sound	text bitmaps other	text import bitmaps	all	text graphics
data model	stack f/bkgnd card field button go to	card filebox link other	guideline button note replacement inquiry	web node link	network frame field link
user interface	card=screen 1 1 scroll text	card=window 1 1 scroll node	1 N scroll	card=window 1 1 scroll node	card=screen 1 1 no scrolling
persistence unit of transfer	card	application?	guideline/node	node	frame
multi-user	no	no?	no/yes	yes	yes
distributed	no	no	no/yes	yes	yes
uniform representation	no	no	limited	no	no
programmable	script XCMD	lisp	no/guidance	no	action
query	no	no	no	no	no
change management	no	no	no	no	no
open data model	mirror	any	no	no	mirror
monolithic vs modular	monolithic	monolithic	monolithic	monolithic	monolithic

Table 1: Comparison of Hypermedia Systems by Basic/Advanced Feature

3.4.2 Relationship of Hypermedia Systems to Near-Miss Systems

A hypermedia reference model must also allow comparison with similar systems that are not usually classified as hypermedia systems. The **big question** is, if we factor these systems into their characteristic dimensions, then how much overlap would there be between systems.[4]

Programming language data structures including *object-oriented programming*, and *AI knowledge representations* including *frame-based systems*, carry data modeling much further than hypermedia systems do today. They provide better uniform representations but have no particular support for foreign objects. In particular, object-oriented programming languages like C++, CLOS, and Smalltalk have common characteristics including object identity, encapsulation, types or classes, and (multiple) inheritance; and they provide procedural attachment. These systems make a strong type-instance distinction and some only allow creation of new data types at compile time.

Persistent programming languages make the data model of the programming language incrementally persistent, managing secondary storage, concurrency, and recovery. *Object-oriented databases* add sets, queries, and indexing; and also change management and schema evolution to persistent languages, but take no particular stand on user interfaces. As such, they generalize *relational database systems*, though implementations of the latter are far more mature. Even more specialized are implementations of *information retrieval* systems which store large text bases persistently, support indexing, but typically provide no editing, data modeling, and only specialized query languages. *Geographic information systems* store graphical information in often-specialized databases.

User interface management systems allow simple user interfaces to be built quickly. *User interface toolkits* like Stanford InterViews and CMU Andrew provide general purpose interface building kits but require programming to put the pieces together. They do not commit to any particular data model. In general, *object libraries* are a way to package up collections of related objects for reuse in building large systems. *Structured graphics editors* can make use of such systems to build generic shapes. Programming language *inspectors* and *class browsers* can be viewed as specialized hypermedia systems for viewing rich representations. *DIRED editors, e-mail previewers, CAD schematic editors*, CASE interfaces, and other semantically specialized graphics editors can browse and edit many views of domain-specific structured data types. *Personal Information Systems* like Symantec GrandView and Lotus Agenda provide many views, including hierarchical views, of simple records via cross indexing.

---

[4] A related implementation question is, are we building almost the same systems over and over without factoring out the common modules?

The kind of objects represented by these systems are usually but not necessarily fine-grained. *Computer-aided publishing (CAP)* systems add primitive objects like text rectangles that may be large and may contain embedded objects. *Text and document markup languages* represent the content of *very rich* hypertext-like systems often specialized to document preparation but also used as the external representations of WYSIWYG document preparation systems like Framemaker. *Syntax-directed structure editors* parse structured text and permit editing, pretty printing, and controlled viewing of programs.

Finally, where *Office Document Architecture* only distinguishes a structural and a page layout architecture for text, graphics, and other static media, technologies like Digital Video Interactive specify how to temporally sequence video and sound and introduce compression.

All of these systems are *almost* hypermedia systems. Some introduce new features including richer data modeling and compression; others seem more like elements of a hypermedia toolkit since they overlap hypermedia systems concentrating only on one basic or advanced feature or another.

3.5 Architecture of an Ideal Hypermedia System

Figure 1 represents an ideal hypermedia system that covers all of the basic and advanced features described earlier in this section. The key point of the architecture is that it is modular and open. This modularity is based on the observations that the functions the modules perform are independent of each other, that is *orthogonality implies modularity.* The only required modules for a basic hypermedia system are the User Interface Toolkit, Domain-specific Data Modeling, Type and Object Manager, and Persistent Storage modules.

Module independence is justified as follows:

- Media types provide primitive representations for text, bitmaps, audio, video, graphics.
- The data model represents structure (nodes, relationships, and content) uniformly. It defines what the hypermedia system can represent. Specializations can define hypermedia objects like card or field or they can define domain-specific objects like transistor or module.

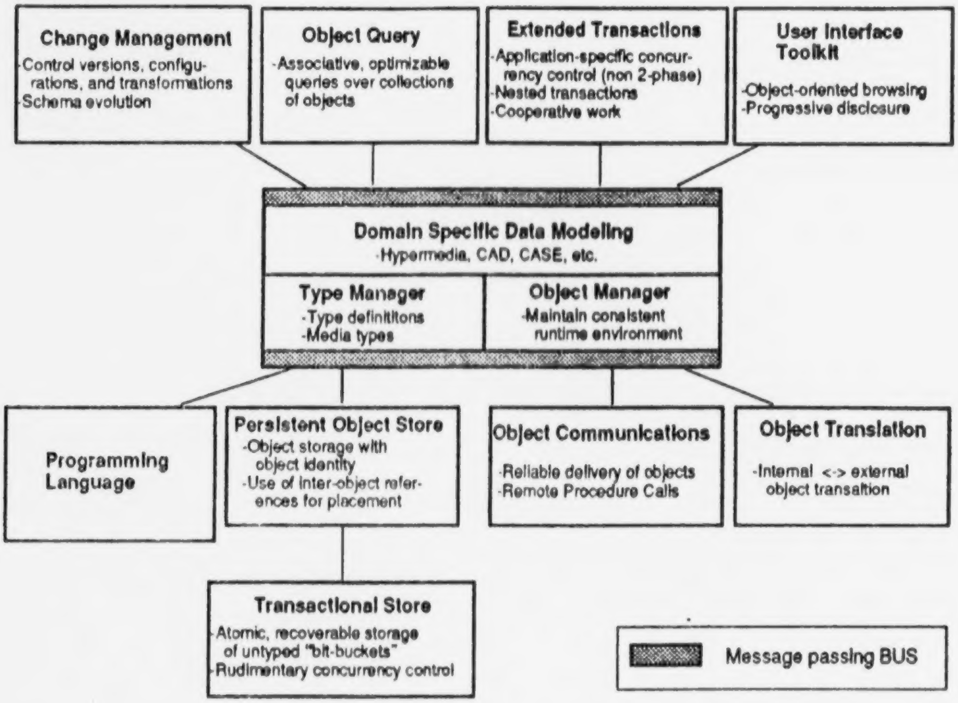

Change Management
-Control versions, configurations, and transformations
-Schema evolution
Object Query
-Associative, optimizable queries over collections of objects
Extended Transactions
-Application-specific concurrency control (non 2-phase)
-Nested transactions
-Cooperative work
User Interface Toolkit
-Object-oriented browsing
-Progressive disclosure
Domain Specific Data Modeling
-Hypermedia, CAD, CASE, etc.
Type Manager
-Type definititons
-Media types
Object Manager
-Maintain consistent runtime environment
Programming Language
Persistent Object Store
-Object storage with object identity
-Use of inter-object references for placement
Object Communications
-Reliable delivery of objects
-Remote Procedure Calls
Object Translation
-Internal <-> external object transaftion
Transactional Store
-Atomic, recoverable storage of untyped "bit-buckets"
-Rudimentary concurrency control
Message passing BUS

Figure 1: Proposed Ideal Hypermedia System Architecture.

- Structure and content can be displayed in many ways (or not at all) so the presentation layer is independent. This can be implemented with a data model-independent user interface toolkit.

- Whether and how this information is mapped to permanent storage is again independent of what is represented, so the storage system is orthogonal. Implement this with an persistent programming language.[5]

- Queries and indexing are related only to whether there are sets, collections, or other navigation paths to iterate through and whether there is cached information (indexes) that can be used to limit the search. Implement this with the open query module of an object-oriented database.

- Systems may or may not version their structure and content. How they do this, if they do, can be studied independently of what they represent, how it is viewed, or whether it is persistent. Implement this with a separate Change Management module.

---

[5] View-independence and Storage-independence from representation are similar to the famous 3-tier model of databases.

- From a single users point of view, whether the system is multi-user or not is largely transparent; the same goes for whether the system is distributed.
- Implement the above functions modularly with well-defined interfaces specified between modules.

3.6 Advantages of this Architecture

A modular, toolkit architecture like the one described in the previous section has these advantages:

- The architecture could be used to build existing hypermedia systems. In that sense, it covers and explains these systems.
- Related systems are implementing several of the modules needed in an ideal hypermedia system. Work on class libraries, persistent languages and OODBs, and user interface toolkits is proceeding in parallel with work on hypermedia systems.
- Since the architecture is modular, modules can be improved individually which would incrementally improve the system. They can be improved by different research groups or vendors. People need not build whole hypermedia systems to experiment with particular parts.
- It will be easier to build the near-miss systems using a modular hypermedia toolkit and the extra capabilities they add to the toolkit will likely benefit existing applications.
- Customized system that only use the modules they need can be constructed.
- If the modules are orthogonal, then consensus that leads to standardization should concentrate on individual abstractions, not portmanteau standards covering many essentially independent parts.

The architecture proposed here is similar to the proposed architecture for *Application Integration Frameworks* being developed by several industrial consortia. These include: USAF

WRDC Engineering Information Systems (EIS), Object Management Group (OMG), CAD Frameworks Initiative (CFI), and CASE Integrated Systems (CIS). As shown in Figure 2, all these efforts provide an object-oriented software backplane architecture into which software services are "plugged." This allows new applications that use the common services of the framework to be built more quickly and to have a "uniform semantics." Applications are simpler to implement since common services are factored out and provided by the framework. To date, framework services include common link protocols like Sun's Link Protocol, help and tutorial services, debugging services, and change management services, all implemented on top of file systems.

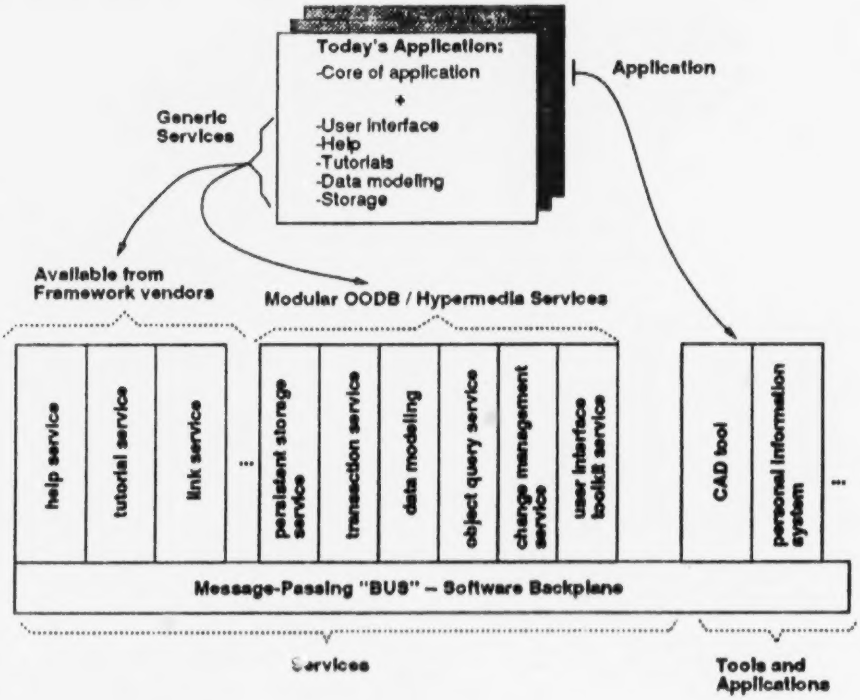


Figure 2: Hypermedia Modules complement Application Integration Frameworks.

Missing ingredients from these framework architectures are the modules offered by a modular OODB, which would permit sharing at the object grain size instead of the file grain-size and querying. Also missing are user interface toolkits and data modeling facilities needed by a hypermedia system. The framework view of the world as modular services fits very well with the proposed modular hypermedia system architecture.

3.7 Conclusion

The reference model presented in this section is incomplete. More work is needed to refine it in many places. Nevertheless, we have shown how it provides a way to compare existing hypermedia systems along orthogonal dimensions and have indicated that it can be extended to relate hypermedia systems to several kinds of near-miss systems. Based on the features of hypermedia systems, an ideal architecture for a hypermedia system was presented and advantages of this architecture were described and related to the architecture of Application Integration Frameworks. An argument was given for how a modular architecture can accelerate progress towards hypermedia standardization.

4 Operational Standards: Where is Consensus Possible?

Operational standards provide means for different *computer systems* to agree to communicate or interface or share. Many sorts are possible in the hypermedia area, reflecting the independent dimensions of the reference model presented earlier. This section identifies some areas where hypermedia standardization might succeed and be useful.

4.1 Common Media Type Representations.

Standards already exist in this area. Appendix A lists some of these. Different media have different properties (linear or 2-D in time or space, discrete or continuous, etc). Conversions among some media representations are algorithmic but lose information (e.g., structured graphics to bitmap, high resolution to low resolution). Often, higher-level structured (or other media) representations are represented in media representations. In some cases we know how to parse the media algorithmically to recognize this information; often we do not.

4.2 Common Hypermedia Abstract Machine and Interchange Format.

The data modeling module of a hypermedia system (including the media types) can be represented equivalently as 1) an *abstract machine* which includes a specification of operations on data (an interpreter) plus an internal representation of the data it can operate on or 2) *an external format* that encodes the application-specific content of the system for storage or transmission.

The Neptune Hypermedia Abstract Machine (HAM) [7] describes a semantic-net abstract machine that includes not only data modeling primitives but also operations for managing change and querying. Representation primitives are nodes, attributes, and values.

By itself, a semantic net data model is so weak that it permits *any* structural information to be encoded. As such, it *represents* very little unless an interpreter looks at the data (at attribute names like *type*). An ASCII linear representation of a semantic net would have the same semantic-less information-bearing properties.

A semantic net representation could be standardized as could an associated linear representation format. The linear format could use Lisp-like parentheses, SGML-like tags, or an easy to parse, hard to understand binary format. But this by itself says nothing about whether hypermedia systems can exchange hypermedia data or cooperate.

4.3 Common Data Model.

The heart of a hypermedia system is the information it can represent. Distinctions like text rectangle, frame, card, field, button, breadcrumb, and so on provide this information. Different hypermedia systems will be able to exchange information only to the extent that there are mappings between their representation primitives. It may often be reasonable to map a font from one system to a different font in another (but not always for all purposes). It may even be reasonable for some purposes to set up mappings from KMS frames to Intermedia nodes to HyperCard cards to Notecards. Similarly, Intermedia links can be mapped to HyperCard fields with simple scripts containing "go to"s. If N hypermedia systems represent the *same* object then a mapping to an intermediate form does not lose information and can be useful.

4 OPERATIONAL STANDARDS: WHERE IS CONSENSUS POSSIBLE?

We often need to perform mappings between different system's representations: if conversion from one system to another is required, we try to map as much information as is useful. In most *instances*, some amount of conversion can happen algorithmically. It is not too interesting that specific content can be moved between hypermedia systems with application-specific mappings. The interesting case involves whether application-independent conversion routine between two systems are useful or possible.

In general, mappings can be one-way (no inverse); they can be non-unique; and they can lose information. All these cases happen in important hypermedia system. Because of the power of scripts, the inverse of mapping Intermedia links to HyperCard fields and go-to scripts is not unique. HyperCard foreground and background cards can be mapped to KMS frames but the "inheritance" is lost. Guide's variable-sized text nodes would need to be mapped to several KMS fixed-size frames. Structured graphics imported into Guide is converted to bitmaps, losing the structure. And so on.

Even when a mapping is established, data exchange between different hypermedia systems will often not preserve the *look and feel* of different hypermedia systems. Thus a Guide node may map to a HyperCard text field but the progressive-disclosure-in-context look and feel of Guide outline processing will be lost.

With all these caveats, it is often useful to build generic conversion programs. PC and Macintish application commonly convert data to their own internal formats, often losing some information. References [13-15] describe systems that explore the problems associated with mapping between different document representations. The Berkeley Vortex system explores how to maintain an incremental, multiple representation mapping between a WYSIWYG editor and a markup language representation.

While it is fruitful to try to define intermediate forms like the Dexter Hypermedia Interchange Format [6] that permit mapping information between today's intermediate forms (since it points out exactly where the mappings cause problems), it seems unlikely that the *behavioral*, script component *so* dominant in HyperCard can be captured without duplicating the entire HyperCard script interpreter in some related Hypermedia system. It may be better to consider whether richer, more uniform representations are better than cards and slots.

4.4 Common Object Libraries.

The X Consortia is considering a standard C++ interface to X-Windows. [13] describes a portable Office Document Architecture toolkit consisting of C subroutines associated with the CMU Andrew Toolkit. Stanford InterViews is a C++ class library implementing a user interface toolkit. It seems likely that we could standardize on C++ libraries in these sorts of area. Such libraries could implement cards, buttons, and so on but could also uniformily implement CAD transisters and layout structures.

4.5 Standard OODBs.

X3/SPARC/DBSSG has recently announced the OODB Task Group which is chartered to assess the potential for standardization in the OODB area. This is especially interesting since many hypermedia researchers look forward to using OODBs to help implement large, shared hypermedia systems. This effort itself may involve several standards: how to seamlessly interface OODBs to various languages to provide persistence and sharing, and how to map data between languages (like Sun's XDR) to allow cross-language sharing.

4.6 Abstract Machines for Querying and Change Management

As mentioned, Neptune HAM defined not only data modeling primitives but also operations for managing change and querying. These are independent dimensions and should be treated as separate abstraction machines. The query engine should define how a set-oriented query engine attaches to a representation, indexes it, and permits powerful queries. A change management abstract machine defines operations on versions, configurations, and transformations.

4.7 Link Protocol

Sun's Link Service and HP New Wave both define a protocol applications can use to set up various kinds of cross-application *links*. HP New Wave appears more powerful in that it would permit cross-application (key-board) macros based on the link service and implement common system-wide protocols for accessing help, tutorials, and other common services. This is an area of potential standardization being covered by the several Frameworks consortia mentioned in section 3.6.

A Hypermedia Standardization Group would complement the Frameworks effort if it concentrated on making some of the services described above available.

5 Conclusions

This paper has provided a reference model for comparing hypermedia systems and an architecture that isolates design decisions to modules. The implication is that we can consider separable subsystems in isolation, then combine the parts to make a whole hypermedia system.

Based on this analysis, several areas where consensus is possible were isolated including: media representations, data model, interchange formats, class libraries (for media types, data modeling types, and domain specific types like CAD), user interface toolkit class libraries, a standard protocol for linking, standards for persistent languages, and abstract machines for queries and change management.

Some of these standards exist, some are being pursued by other official or de facto standards bodies, and some are new possibilities. While it seems too early to consider standardizing today's hypermedia systems with their several limitations, the effort toward building consensus is helping us to understand these systems better and to identify potential areas where standards can help.

6 Appendix A: Related Standards and Common Formats

This section lists some common external representations of information used for various purposes. It is included since it represents a beginning of a section on related standards. It also demonstrated some of the breadth of kinds of objects that hypermedia systems will need to represent.

- communication protocols
  - SCSI -- Small Computer Systems Interface
- external representations for data structures
  - XDR -- Sun's external data representation
- device-independent procedural page/screen description formats
  - DVI -- for TEX
  - ditroff -- for troff
  - imPRESS(TM) -- document for printing on an IMAGEN laser printer
  - EPS -- Encapsulated Postscript -- generated by Adobe Illustrator(TM), Cricketdraw(TM), Aldus Freehand(TM) on the Macintosh and Media Logic's Artisan(TM) on the Sun; also Display Postscript and color versions
- media type interchange formats (specific ''document contents'' like characters, raster graphics, geometric graphics, sound, video, etc). Note: Several of these representations represent structure and content.
  - ASCII - text
  - DIF -- Document Interchange Format -- used to interchange text and formatting instructions across a wide variety of wordprocessors and publishing systems
  - troff - the standard Unix text processing utility
  - DCA -- IBM's Revisable Form Text Document Content Architecture. Many popular word processors can store documents in this format

(including IBM Displaywriter(R), WordPerfect(R), Wang(R), MultiMate(TM), Wordstar2000(R), Samna IV (TM), OfficeWriter(R), and MicroSoft Word(R) can store documents in this format. Does not support graphics.

Scribe

Tex, LaTex -- popular text formatting language, weak on non-textual objects, primitives for tables

MIF -- Framemaker's Maker Interchange Format

Interleaf, Microsoft Word, HyperCard, WordStar, Ventura, ... many products provide a way to save and restore their state.

EDA/VGA/CGA -- bitmap screen sizes/resolutions on different PCs

X3H3 GKSM -- Graphical Kernal System Metafile (polyline, polymarker, text, fill area, cell array, generalized drawing primitive) (A second metafile standard provides a way to encode a sq sequence of GKS commands. The description of the objects, not the image is saved.

PHIGS --

GIF -- graphic interchange format

ISO Computer Graphics Metafile --

PICT -- Macintosh standard graphics description format

pic -- a language for typesetting graphics

HPGL -- a popular plotter output format used by many workstation CAD programs like AutoCAD

IGES -- a standard graphics interchange format used by many workstation CAD programs

MacDraw - Macintosh(TM) MacDraw files--QuickDraw--toolbox ROM routines

NTSC -- U.S. etc television format standard for production and transmission; Europe uses PAL; HDTV and ACTV are next generation

  - SMPTE -- Society of Motion Picture and Television Engineers--time code for syncing audio, video, film
- document/audio-video representation and interchange formats
  - SGML -- ANSI/ISO Standard Generalized Markup Language. Uses markups (tags) to create an indirections between intent and rendering. Does not support graphics.
  - ODIF -- Office Document Interchange Format. ODA distinguishes a logical hierarchy and a layout hierarchy
  - CD-I -- Compact Disk Interactive, compression/decompression formats
  - DVI -- Digital Video Interative. Text, audio, video stills, and video motion, at various resolutions, mixed, compression/decompression formats
- cad-specific interchange formats
  - EDIF -- Electronic Data Interchange Format
  - VHDL -- VHSIC Hardware Description Language
  - CIF -- Caltech Interchange Format
- product interchange format
  - PDES -- Product Data Exchange Specification
  - EDI -- Electronic Data Interchange

7 Appendix B: Document Log

The document log lists bibliographies, conference proceedings, key papers, and other documents that are related to the hypermedia standardization effort.

[1] Jakob Nielsen, "Hypertext Bibliography," Hypermedia, Taylor Graham (ed), 1:1, 1989. This bibliography references key papers by Bush, Engelbart, Kay, and Nelson; surveys and books by Conklin and Schneiderman; systems like Intermedia, Neptune, KMS, HyperCard, Notecards, Guide, Object Lens; and other technical papers on hypermedia.

[2] Proceeding of the ACM SIGPLAN/SIGOA Symposium on Text Manipulation, Portland,

Oregon, June 8-10 1981. Available as SIGPLAN Notices 16(6) or SIGOA Newsletter 2(1-2).

[3] Hypertext'87 Proceedings, ACM press, Chapel Hill, NC, November 13-15, 1987.

[4] Hypertext'89 Proceedings, ACM press, Pittsburgh, November 5-8, 1989.

[5] ACM Conference on Document Processing Systems, ACM Press, Santa Fe, New Mexico, December 5-9, 1989.

[6] Bornstein, Jeremy, Frank Halasz, and Tim Oren. "Dexter Hypertext Interchange Format (DHIF)-Discussion and Format Specification--version 1.4", unpublished, November 3, 1989.

[7] Campbell, B. and J. M. Goodman. "HAM: A General Purpose Hypertext Abstract Machine," Communications of the ACM, 31:7, July, 1988.

[8] IBM (1983). Document Content Architecture: Revisable-Form-Text Reference. SC23-0758.

[9] International Organization for Standardization (1986). Standard Generalized Markup Language. ISO DIS 8879.

[10] International Organization for Standardization (1986). Computer Graphics Metafile. ISO IS 8632.

[11] International Organization for Standardization (1987). Office Document Architecture. ISO DIS 8613.

[12] Knoerdel, J. and Ward Watkins, S. (1984) Document Interchange Format. National Bureau of Standards, NBSIR 84-2836.

[13] Sherman, Mark. "Experiences Interchanging Multimedia Documents using ODA," Conference on New Horizons in Electronic Media, International Telecommunications Union, October 4-7, 1989, Geneva, Switzerland, pp 429-433.

[14] de La Beaujardiere, Jean-Marie, "Well-Established Document Interchange Formats," Document Manipulation and Typography, J C. van Vliet (ed), Cambridge University Press, 1988.

[15] S Mamrak, M. Kaelbling, C. Nicholas, and M. Share. "Chameleon: A System for Solving the Data Translation Problem." TR24, Department of Computer and Information Science, The Ohio State University, August, 1988.

APPENDICES

Hypermedia Bibliography, 1989

Paul Kahn,
Institute for Research in Information and Scholarship
Brown University, Box 1946
Providence RI 02912

Since the last time we compiled this bibliography in November 1987 for the Hypertext '87 Workshop, there has been an explosion of hypertext literature. When we started the bibliography project at IRIS in 1983, we thought it would be possible to collect every book, conference paper and journal article on the subject of hypertext. In 1989, that seems an impossible goal. We hope our collection includes a large portion of the current literature, but every day we learn of new papers that are not part of our collection.

This version, prepared for distribution by NIST, contains only references to material we have been able to collect over the past six years. The reference list differs substantially from the 1987 version. In 1987 there just were not that many papers focused entirely on hypertext, so we included in the bibliography many papers that, while only tangentially related to the topic of hypertext, had been influential in helping us think about the subject. Now that there are so many papers focused solely on hypertext, we have opted to narrow the scope of the bibliography and include only those references that are exactly on the topic.

A longer version of this bibliography, containing the following list plus an annotated list of selected sources is available for $3.00 from IRIS (Brown University, Box 1946, Providence RI 02912).

This bibliography represents a collaborative effort of not only members of the IRIS staff, but also of a number of others who have worked on compiling bibliographies, most notably John Leggett (Texas A&M), Jakob Nielson (Technical University of Denmark), and Rosemary Simpson (Boston Computer Society).

The list of references below is arranged alphabetically by first author.

Agosti, Maristella. "Is Hypertext A New Model of Information Retrieval?" *Proceedings of the 12th International Online Information Meeting.* December 6-8, 1988, London, England. New Jersey: Learned Information, 1988. 57-62.

Akscyn, Robert M., Donald L. McCracken and Elise A. Yoder. "KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations." *Communications of the ACM*, Vol. 31, No. 7 (July 1988): 820-835.

Akscyn, Robert M. and Donald L. McCracken. "The ZOG Approach to Database Management." *Proceedings of the Trends and Applications Conference: Making Databases Work.* Gaithersburg, MD, May, 1984.

Alexander, George. "Knowledge Management Systems from Scribe: Hypertext for Groups." *The Seybold Report on Publishing Systems*, Vol. 18, No. 12 (1989): 11-17.

Allen, Todd, Robert Nix and Alan Perlis. "PEN: A Hierarchical Document Editor." *Proceedings of the ACM SIGPLAN/SIGOA Conference on Text Manipulation.* Portland, Oregon, June, 1981.

Allinson, Lesley and Nick Hammond. "A Learning Support Environment: The Hitch Hikers Guide." in *Hypertext: Theory into Practice*, Ray McAleese, (editor). Norwood, NJ: Ablex Publishing Corporation, 1989. 62-74.

Alschuler, Liora. "Hand-Crafted Hypertext-Lessons from the ACM Experiment." in *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*, Edward Barrett, (editor). Cambridge, MA: The MIT Press, 1989. 343-361.

Ambron, Sueann and Kristina Hooper. *Interactive Multimedia.* Redmond, WA: Microsoft Press, 1988.

Backer, D. and Stephen Gano. "Dynamically Alterable Videodisk Displays." *Proceedings of Graphics Interface 82.* Toronto, Canada, May 1982.

Baird, Patricia and Mark Percival. "Glasgow On-Line: Database Development using Apple's HyperCard." in *Hypertext: Theory into Practice*, Ray McAleese, (editor). Norwood, NJ: Ablex Publishing Corporation, 1989. 75-92.

Barrett, Edward. *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information.* Cambridge, MA: The MIT Press, 1989.

Baskin, A. B. "Logic Nets: Variable-Valued Logic Plus Semantic Networks." *International Journal on Policy Analysis and Information Systems*, Vol. 4 (1980): 269.

Beeman, William O., Kenneth T. Anderson, Gail Bader, James Larkin, Anne P. McClard, Patrick J. McQuillan and Mark Shields. "Hypertext and Pluralism: From Lineal to Non-lineal Thinking." *Hypertext '87 Papers*. November 13-15, 1987, Chapel Hill, NC. New York: ACM, 1989. 67-88.

Beeman, William O., Kenneth T. Anderson, Gail Bader, James Larkin, Anne P. McClard, Patrick McQuillan and Mark Shields. *Intermedia: A Case Study of Innovation in Higher Education. Final Report to the Annenberg/CPB Project*, IRIS, Brown University, Providence, RI, 1988.

Begeman, Michael L. and Jeff Conklin. "The Right Tool for the Job." *Byte*, Vol. 12, No. 10 (October 1988): 255-266.

Begeman, Michael L., P. Cook, Clarence Ellis, M. Graf, G. Rein and T. Smith. "PROJECT NICK: Meetings Augmentation and Analysis." *Computer-Supported Cooperative Work (CSCW '86) Proceedings*. December 3-5, Austin, TX, 1986.

Bernstein, Mark. "The Bookmark and the Compass: Orientation Tools for Hypertext Users," *ACM SIGOIS Bulletin*. Robert B. Allen, (editor). Vol. 9, No. 4 (October 1988): 34-45.

Bender, Walter. "Imaging and Interactivity." *Fifteenth Joint Conference on Image Technology*. November, Tokyo, Japan, 1984.

Bernstein, Mark (editor). *AI and Hypertext: Issues and Directions*. AAAI-88 Workshop proceedings, August 1988, St. Paul, MN, Watertown, MA: Eastgate Systems, Inc., 1988.

Bhargava, Hemant, Michael Bieber and Steven O. Kimbrough. "OONA, MAX and the WYWWYWI Principle: Generalized Hypertext and Model Management in a Symbolic Programming Environment." *Proceedings of ICIS* '88. 179-191.

Bieber, Michael and Steven O. Kimbrough. *On Generalizing the Concept of Hypertext*, Technical Report BCCS-89-03, Computer Science Department, Boston College, Chestnut Hill, MA, September 1989.

Bigelow, James and Victor Riley. "Manipulating Source Code in Dynamic Design." *Hypertext '87 Papers*. November 13-15, 1987, Chapel Hill, NC. New York: ACM, 1989. 397-408.

Biggerstaff, Ted, Clarence Ellis, Frank G. Halasz, C. Kellogg, C. Richter and D. Webster. "Information Management Challenges in the Software Design Process." *Database Engineering*, Vol. 10, No. 1 (March, 1987): 24-31.

Binder, Carl. "The Promise of a Paperless Workplace." *Optical Insights*, (Fall 1987).

Binder, Carl. "The Window Book Technology." *Boston Computer Society Training and Documentation Newsletter*, (Fall 1986).

Björklund, Lisbeth, Birgitta Olander and Linda C. Smith. "The Personal Hypercatalog." *Annual Meeting of the American Society for Information Science*. October 30-November 1, 1989, Washington, DC, 1989.

Blair, David C. and M. E. Maron. "An Evaluation of Retrieval Effectiveness for a Full-Text Document-Retrieval System." *Communications of the ACM*, Vol. 28, No. 3 (March 1985): 289-299.

Bolt, Richard A. *Spatial Data-Management, DARPA Report*, MIT Architecture Machine Group, Cambridge, MA, 1979.

Bolter, Jay David and Michael Joyce. "Hypertext and Creative Writing." *Hypertext '87 Papers*. November 13-15, 1987, Chapel Hill, NC. New York: ACM, 1989. 41-50.

Bourne, John R., Jeff Cantwell, Authur J. Brodersen, Brian Antao, Antonis Koussis and Yen-Chun Huang. "Intelligent Hypertutoring in Engineering." *Academic Computing*, (September 1989): 18-20, 42-48.

Bovey, J. D. and Peter J. Brown. "Interactive Document Display and its Use in Information Retrieval." *Journal of Documentation*, Vol. 43, No. 2 (June 1987): 125-137.

Brockmann, R. John, William Horton and Keven Brock. "Limited Freedom: Linear Reflections on Nonlinear Texts." in *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*, Edward Barrett, (editor). Cambridge, MA: The MIT Press, 1989. 162-205.

Brown, John Seely. *Notes Concerning Design Functionality, Issues and Philosophy for an AuthoringLand*, Xerox Palo Alto Research Center, Palo Alto, CA, February 1982.

Brown, John Seely. "Process versus Product: A Perspective on Tools for Communal and Informal Electronic Learning." in *Education in the Electronic Age: A Report from the Learning Lab*, WNET/Thirteen Learning Lab. New York: WNET, 1983. 41-58.

Brown, Peter J. "Interactive Documentation." *Software–Practice and Experience*, Vol. 16, No. 3 (March 1986): 291-299.

Brown, Peter J. "A Simple Mechanism for the Authorship of Dynamic Documents." in *Text Processing and Document Manipulation: Proceedings of the International Conference*, J. C. van Vliet, (editor). Cambridge: Cambridge University Press, 1986. 34-42.

Brown, Peter J. "Viewing Documents on a Screen." in *CD-ROM: The New Papyrus*, Steve Lambert and Suzanne Ropiequet, (editors). Redmond, WA: Microsoft Press, 1986. 175-186.

Brown, Peter J. "On-Line Documentation." in *State of the Art in Computer Graphics*, Earnshaw, (editor). Springer-Verlag, 1987.

Brown, Peter J. "Turning Ideas into Products: The Guide System." *Hypertext '87 Papers*. November 13-15, 1987, Chapel Hill, NC. New York: ACM, 1989. 33-40.

Brown, Peter J. "Hypertext: The Way Forward." in *Document Manipulation and Typography*, J. C. van Vliet, (editor). Cambridge: Cambridge University Press, 1988. 183-191.

Brown, Peter J. "Linking and Searching in Hypertext." *EP-odd*, Vol. 1, No. 1 (1988): 45-53.

Buchert, R. F., K. H. Evers and P. R. Santucci. "SADT/Saint Simulation Technique." *National Aerospace and Electronics Conference Proceedings*. 1981,

Bush, Vannevar. "As We May Think." *Atlantic Monthly*, Vol. 176, No. 1 (July 1945): 101-108.

Bush, Vannevar. "Memex Revisited." in *Science Is Not Enough* by Vannevar Bush. New York: William Morrow, 1967. 75-101.

Campbell, Brad and Joseph M. Goodman. "HAM: A General Purpose Hypertext Abstract Machine." *Communications of the ACM*, Vol. 31, No. 7 (July 1988): 856-861.

Carlson, Patricia Ann. "Hypertext and Intelligent Interfaces for Text Retrieval." in *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*, Edward Barrett, (editor). Cambridge, MA: The MIT Press, 1989. 59-76.

Carmody, Steve, W. Gross, Theodor H. Nelson, David E. Rice and Andries van Dam. "A Hypertext Editing System for the /360." in *Pertinent Concepts in Computer Graphics*, M. Faiman and J. Nievergelt, (editors). University of Illinois Press, 1969. 63-88.

Carr, C. "Hypertext: A New Training Tool?" *Educational Technology*, Vol. 28, No. 8 (1988): 7-11.

Carroll, John M. and Amy P. Aaronson. "Learning by Doing with Simulated Intelligent Help." in *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*, Edward Barrett, (editor). Cambridge, MA: The MIT Press, 1989. 423-452.

Cashin, P., M. Robinson and D. Yates. "Experience with SCRAPBOOK, A Non-Formatted Data Base System." *Proceedings IFIPS Congress*, 1973.

Catano, James V. "Poetry and Computers: Experimenting with the Communal Text." *Computers and the Humanities*, Vol. 13 (1979): 269-275.

Catlin, Timothy, Paulette E. Bush and Nicole Yankelovich. "InterNote: Extending a Hypermedia Framework to Support Annotative Collaboration." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 365-378.

Catlin, Timothy J. O. and Karen E. Smith. "Anchors for Shifting Tides: Designing a 'Seaworthy' Hypermedia System." *Proceedings of the 12th International Online Information Meeting*. December 6-8, 1988, London, England. Oxford and New Jersey: Learned Information, 1988. 15-25.

Charney, Davida. "Comprehending Non-Linear Text: The Role of Discourse Cues and Reading Strategies." *Hypertext '87 Papers*. November 13-15, 1987, Chapel Hill, NC. New York: ACM, 1989. 109-120.

Charney, Davida and Lynne M. Reder. "Designing Interactive Tutorials for Computer Users." *Human-Computer Interaction*, Vol. 2, No. 4 (1986): 297-317.

Chignell, Mark H. and Richard M. Lacy. "Project Jefferson: Integrating Research and Instruction." *Academic Computing*, (September 1988): 12-17, 40.

Christodoulakis, Stavros and Stephan Graham. "Browsing Within Time-Driven Multimedia Documents." *Conference on Office Information Systems*. Robert B. Allen, (editor). March 23-25, 1988, Palo Alto, CA. New York: ACM, 1988. 219-227.

Claassen, W. T. and T. J. D. Bothma. "Structuring Diverse Types of Information in Hypertext: The Case of Biblical Information." *Proceedings of the 12th International Online Information Meeting*. December 6-8, 1988, London, England. Oxford and New Jersey: Learned Information, 1988. 83-90.

Clitherow, Peter, Doug Riecken and Michael Muller. "VISCAR: A System for Inference and Navigation of Hypertext." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 293-304.

Collier, George H. "Thoth-II: Hypertext with Explicit Semantics." *Hypertext '87 Papers*. November 13-15, 1987, Chapel Hill, NC. New York: ACM, 1989. 269-290.

Combelic, D. "User Experience with New Software Methods (SADT)." *Proceedings of the National Computer Conference*, 1978. 631-633.

Conklin, Jeff. *A Survey of Hypertext*, MCC Technical Report STP-356-86, Rev. 2. MCC Software Technology Program, Austin, TX, December 3, 1986.

Conklin, Jeff. "Hypertext: An Introduction and Survey." *IEEE Computer*, Vol. 20, No. 9 (September, 1987): 17-41.

Conklin, Jeff and Michael L. Begeman. "gIBIS: A Hypertext Tool for Team Design Deliberation." *Hypertext '87 Papers*. November 13-15, 1987, Chapel Hill, NC. New York: ACM, 1989. 247-252.

Conklin, Jeff and Michael Begeman. "gIBIS: A Tool for All Reasons." *Journal of American Society for Information Science*, Vol. 40, No. 3 (May 1989): 200-213.

Consens, Mariano P. and Alberto O. Mendelzon. "Expressing Structural Hypertext Queries in GraphLog." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 269-292.

Cooke, Peter and Ian Williams. "Design Issues in Large Hypertext Systems for Technical Documentation." in *Hypertext: Theory into Practice*, Ray McAleese, (editor). Norwood, NJ: Ablex Publishing Corporation, 1989. 93-104.

Corda, U. and G. Facchetti. "Concept Browser: A System for Interactive Creation of Dynamic Documentation." in *Text Processing and Document Manipulation: Proceedings of the International Conference*, J. C. van Vliet, (editor). Cambridge: Cambridge University Press, 1986.

Crane, Gregory. "From the Old to the New: Integrating Hypertexts into Traditional Scholarship." *Hypertext '87 Papers*. November 13-15, 1987, Chapel Hill, NC. New York: ACM, 1989. 51-56.

Croft, W. Bruce and Howard Turtle. "A Retrieval Model Incorporating Hypertext Links." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 213-224.

Crouch, Donald B., Carolyn J. Crouch and Glenn Andreas. "The Use of Cluster Hierarchies in Hypertext Information Retrieval." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 225-238.

Dede, Christopher J. "Empowering Environments, Hypermedia, and Microworlds." *The Computing Teacher*, Vol. 15, No. 3 (November 1987): 20-26.

Delisle, Norman and Mayer Schwartz. "Contexts — A Partitioning Concept for Hypertext." *Computer-Supported Cooperative Work (CSCW '86) Proceedings*. December 3-5, Austin, TX, 1986. 147-152.

Delisle, Norman and Mayer Schwartz. *Neptune: A Hypertext System for CAD Applications*, CR-85-50. Tektronix Computer Research Laboratory, Beaverton, OR, January 1986.

DeRose, Steven J. "Expanding the Notion of Links." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 249-258.

DeYoung, Laura. "Hypertext Challenges in the Auditing Domain." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 169-180.

diSessa, Andrea A. "A Principled Design for an Integrated Computational Environment." *Human-Computer Interaction*, Vol. 1 (1985): 1-47.

diSessa, Andrea A. and Harold Abelson. "Boxer: A Reconstructable Computational Medium." *Communications of the ACM*, Vol. 29, No. 9 (September, 1986): 859-868.

Doland, Virginia M. "The Hermeneutics of Hypertext." *Proceedings of the 12th International Online Information Meeting.* December 6-8, 1988, London, England. Oxford and New Jersey: Learned Information, 1988. 75-82.

Doland, Virginia M. "Hypermedia as an Interpretive Act." *Hypermedia*, Vol. 1, No. 1 (Spring 1989): 6-19.

Duffy, Thomas M., Brad Mehlenbacher and Jim Palmer. "The Evaluation of Online Help Systems: A Conceptual Model." in *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*, Edward Barrett, (editor). Cambridge, MA: The MIT Press, 1989. 362-387.

Duncan, Elizabeth B. "Structuring Knowledge Bases for Designers of Learning Materials." *Hypermedia*, Vol. 1, No. 1 (Spring 1989): 20-33.

Duncan, Elizabeth B. "A Faceted Approach to Hypertext?" in *Hypertext: Theory into Practice*, Ray McAleese, (editor). Norwood, NJ: Ablex Publishing Corporation, 1989. 157-163.

Edwards, Deborah M. and Lynda Hardman. "'Lost in Hyperspace': Cognitive Mapping and Navigation in a Hypertext Environment." in *Hypertext: Theory into Practice*, Ray McAleese, (editor). Norwood, NJ: Ablex Publishing Corporation, 1989. 105-125.

Egan, Dennis E., Joel R. Remde, Thomas K. Landauer, Carol C. Lockbaum and Louis M. Gomez. "Behavioral Evaluation and Analysis of a Hypertext Browser." *Proceedings of the Annual Meeting of the American Educational Research Association.* April 30-May 4, 1989, Austin, TX. 205-210.

Egan, Dennis E., Joel R. Remde, Louis M. Gomez, Thomas K. Landauer, Jennifer Eberhardt and Carol C. Lochbaum. "Formative Design Evaluation of SuperBook." *ACM Transactions on Information Systems*, Vol. 7, No. 1 (January 1989): 30-57.

Ehrlich, K. and Janet H. Walker. "High Functionality, Information Retrieval, and the Document Examiner." in *Personalized Intelligent Information Systems, Report on a Workshop (Breckenridge, CO)*, Fischer, G. and H. Nieper, (editors). 1987.

Engelbart, Douglas C. "A Conceptual Framework for the Augmentation of Man's Intellect." in *Vistas in Information Handling, Volume 1*, P. D. Howerton and D. C. Weeks, (editors). Washington, D.C.: Spartan Books, 1963. 1-29.

Engelbart, Douglas C. "Coordinated Information Services for a Discipline or Mission-Oriented Community." *Second Annual Computer Communications Conference.* San Jose, CA, January, 1973.

Engelbart, Douglas C. "Design Considerations for Knowledge Workshop Terminals." *AFIPS Conference Proceedings - 1973 National Computer Conference and Exposition.* June 4-8, 1987, New York, NY. Montvale, NJ: AFIPS Press, 1973. 221-227.

Engelbart, Douglas C. "Toward Integrated Evolutionary Office Automation Systems." *Proceedings of the International Engineering Management Conference.* October 16-18, Denver, CO, 1978.

Engelbart, Douglas C. "Evolving the Organization of the Future: A Point of View." *Emerging Office Systems.* Robert M. Landau and James H. Blair, (editors), 1982. 287-297.

Engelbart, Douglas C. "Authorship Provisions in Augment." *Proceedings of the 1984 COMPCON Conference (COMPCON '84 Digest).* February 27-March 1, 1984, San Francisco, CA. IEEE Computer Society Press, Spring, 1984. 465-472.

Engelbart, Douglas C. "Collaboration Support Provisions in Augment." *Proceedings of the AFIPS Office Automation Conference (OAC '84 Digest).* February, 1984, Los Angeles, CA, 1984. 51-58.

Engelbart, Douglas C. and William K. English. "A Research Center for Augmenting Human Intellect." *AFIPS Conference Proceedings, 1968 Fall Joint Computer Conference.* December 9-11, 1968, San Francisco, CA. Montvale, NJ: AFIPS Press, Fall 1968. 395-410.

Engelbart, Douglas C. with Kristina Hooper. "The Augmentation System Framework." in *Interactive Multimedia*, Sueann Ambron and Kristina Hooper, (editors). Redmond, WA: Microsoft Press, 1988. 15-32.

Engelbart, Douglas C., Richard W. Watson and James C. Norton. "The Augmented Knowledge Workshop." *AFIPS Conference Proceedings, 1973 National Computer Conference and Exposition.* June 4-8, 1973, New York, NY. Montvale, NJ: AFIPS Press, 1973. 9-21.

English, William K., Douglas C. Engelbart and M. L. Berman. "Display-Selection Techniques for Text Manipulation." *IEEE Transactions on Human Factors and Electronics*, Vol. 8, No. 1 (March 1967): 5-15.

Evenson, Shelly, John Rheinfrank, Fitch Richardsonsmith and Wendy Wulff. "Towards a Design Language for Representing Hypermedia Cues." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 83-92.

Fairchild, Kim F., Steve E. Poltrock and George W. Furnas. "SemNet: Three-dimensional Graphic Representations of Large Knowledge Bases." in *Cognitive Science and its Applications for Human-Computer Interaction*, R. Guindon, (editor). Hillsdale, NJ: Lawrence Erlbaum Associates, in press.

Feiner, Steven. "Interactive Documents." in *Design in the Information Environment*, P. Whitney and C. Kent, (editors). New York: Alfred Knopf, 1985. 118-132.

Feiner, Steven. "Seeing the Forest for the Trees: Hierarchical Display of Hypertext Structure." *Conference on Office Information Systems*. March 23-25, 1988, Palo Alto, CA. New York: ACM. 205-212.

Feiner, Steven, Sandor Nagy and Andries van Dam. "An Experimental System for Creating and Presenting Interactive Graphical Documents." *ACM Transactions on Graphics*, Vol. 1, No. 1 (January 1982): 59-77.

Feiner, Steven, Sandor Nagy and Andries van Dam. "An Integrated System for Creating and Presenting Complex Computer-Based Documents." *Computer Graphics*, Vol. 15, No. 3 (August 1981): 181-189.

Feiner, Steven, Sandor Nagy and Andries van Dam. "Online Documents Combining Pictures and Texts." *Proceedings of the International Conference on Research Trends in Document Preparation Systems*. February 27-28, Lausanne, Switzerland. Lausanne and Zurich: Swiss Institutes of Technology, 1981. 1-4.

Fischer, Gerhard, Raymond McCall and Anders Morch. "JANUS: Integrating Hypertext with a Knowledge-Based Design Environment." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 105-118.

Fish, Robert S., Robert E. Kraut, Mary D. P. Leland and Michael Cohen. "Quilt: A Collaborative Tool for Cooperative Writing." *ACM SIGOIS Bulletin*. Robert B. Allen, (editor). (March 1988): 30-37.

Foss, Carolyn L. "Effective Browsing in Hypertext Systems." *Proceedings of the Conference on User-Oriented Content-Based Text and Image Handling (RIAO 88)*. March 21-24, MIT, Cambridge MA. Centre de Hautes Etudes Internationales d'Informatique Documentaire, 1988. 83-98.

Foster, Edward. "Outliners: A New Way of Thinking." *Personal Computing*, (May, 1985): 74.

Foster, Gregg and Mark Stefik. "Cognoter, Theory and Practice of a Collaborative Tool." *Computer-Supported Cooperative Work (CSCW '86) Proceedings*. December 3-5, Austin, TX, 1986. 7-15.

Frisse, Mark. "From Text to Hypertext." *Byte*, (October 1988): 247-253.

Frisse, Mark E. "Searching for Information in a Hypertext Medical Handbook." *Communications of the ACM*, Vol. 31, No. 7 (July 1988): 880-886.

Frisse, Mark E. and Steve B. Cousins. "Information Retrieval from Hypertext: Update on the Dynamic Medical Handbook Project." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 199-212.

Furuta, Richard and P. David Stotts. "Programmable Browsing Semantics in Trellis." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 27-42.

Garg, Pankaj K. "Abstraction Mechanisms in Hypertext." *Communications of the ACM*, Vol. 31, No. 7 (July 1988): 862-870.

Garg, Pankaj K. and Walt Scacchi. "Composition of Hypertext Nodes." *Proceedings of the 12th International Online Information Meeting*. December 6-8, 1988, London, England. Oxford and New Jersey: Learned Information, 1988. 63-73.

Garg, Pankaj K. and Walt Scacchi. "A Hypertext System to Manage Software Life Cycle Documents." *21st Hawaii International Conference on Systems*. Honolulu HI, 1987.

Garg, Pankaj K. and Walt Scacchi. "On Designing Intelligent Hypertext Systems for Information Management in Software Engineering." *Hypertext '87 Papers*. November 13-15, 1987, Chapel Hill, NC. New York: ACM, 1989. 409-432.

Garrett, L. Nancy and Karen E. Smith. "Building a Timeline Editor from Prefab Parts: The Architecture of an Object-oriented Application." *Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA '86)*. September 29-October 2, Portland, Oregon,1986. 202-213.

Garrett, L. Nancy, Karen E. Smith and Norman Meyrowitz. "Intermedia: Issues, Strategies, and Tactics in the Design of a Hypermedia Document System." *Computer-Supported Cooperative Work (CSCW '86) Proceedings*. December 3-5, Austin, TX, 1986. 163-174.

Gaulding, Jill and Boris Katz. "Using 'Word-Knowledge' Reasoning for Question Answering." in *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*, Edward Barrett, (editor). Cambridge, MA: The MIT Press, 1989. 403-422.

Glushko, Robert J. "Design Issues for Multi-Document Hypertexts." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 51-60.

Glushko, Robert J., M. D. Weaver, T. A. Coonan and J. E. Lincoln. "Hypertext Engineering: Practical Methods for Creating a Compact Disc Encyclopedia." *Proceedings of the ACM Conference on Document Processing Systems*. December 5-9, 1988, Santa Fe, New Mexico. New York: ACM, 1988. 11-19.

Goodman, Danny. *The Complete HyperCard Handbook*. New York: Bantam Books, 1987.

Greenes, Robert A. "Knowledge Management as an Aid to Medical Decision Making and Education: The Explorer-1 System." *Proceedings MEDINFO '86*. Elsevier Science Publishers B.V., 1986. 895-899.

Greenes, Robert A. "Toward More Effective Radiologic Consultation: Design of a Desktop Workstation to Aid in the Selection and Interpretation of Diagnostic Procedures." *Proceedings Eighth Conference on Computer Applications in Radiology*. May 1984, St. Louis, MO. 553-562.

Gregory, Roger. "Xanadu—Hypertext from the Future." *Dr. Dobb's Journal*, No. 75 (January, 1983): 28-35.

Grice, Roger A. "Online Information: What Do People Want? What Do People Need?" in *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*, Edward Barrett, (editor). Cambridge, MA: The MIT Press, 1989. 22-44.

Gullichsen, Eric, D. D'Souza, P. Lincoln and T. Casey. *The PlaneTextBook*, STP-333-86(P). MCC Software Technology Program, Austin, TX, 1986.

Halasz, Frank G. "NoteCards: A Multimedia Idea Processing Environment." in *Interactive Multimedia*, Sueann Ambron and Kristina Hooper, (editors). Redmond, WA: Microsoft Press, 1988. 105-110.

Halasz, Frank G. "Reflections on Notecards: Seven Issues for the Next Generation of Hypermedia Systems." *Communications of the ACM*, Vol. 31, No. 7 (July 1988): 836-855.

Halasz, Frank G., Thomas P. Moran and Randall H. Trigg. "NoteCards in a Nutshell." *Proceedings of the CHI and GI '87 Conference on Human Factors in Computing Systems*. J. M. Carroll and P. P. Tanner, (editors). April 1987, Toronto. New York: ACM, 1987. 45-52.

Hammwohner, Rainer and Ulrich Thiel. "Content-Oriented Relations Between Text Units—A Structural Model for Hypertexts." *Hypertext '87 Papers*. November 13-15, 1987, Chapel Hill, NC. New York: ACM, 1989. 155-174.

Hardman, Lynda. "Evaluating the Usability of the Glasgow Online Hypertext." *Hypermedia*, Vol. 1, No. 1 (Spring 1989): 34-63.

Harland, J.S. "Human Factors Engineering and Interface Development: A Hypertext Tool Aiding Prototyping Activity." in *Hypertext: Theory into Practice*, Ray McAleese, (editor). Norwood, NJ: Ablex Publishing Corporation, 1989. 126-137.

Harvey, Greg. *Understanding HyperCard*. Alameda, CA: SYBEX, Inc., 1988.

Hayes, Phil and Jeff Pepper. "Towards an Integrated Maintenance Advisor." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 119-128.

Herot, C., R. Carling, M. Friedell and D. Kramlich. "A Prototype Spatial Data Management System." *Computer Graphics*, Vol. 14, No. 3 (July 1980): 63-70.

Herrstrom, David S. and David G. Massey. "Hypertext in Context." in *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*, Edward Barrett, (editor). Cambridge, MA: The MIT Press, 1989. 45-58.

Hershey, William. "Software Review: Idea Processors." *Byte*, Vol. 10, No. 6 (June, 1985): 337-350.

Hiltz, Starr Roxanne. "The 'Virtual Classroom': Using Computer-Mediated Communication for University Teaching." *Journal of Communication*, (Spring, 1968): 95-104.

Hiltz, Starr Roxanne and Murray Turoff. *The Network Nation: Human Communication via Computer*. Reading, MA: Addison-Wesley Publishing Company, Inc., 1978.

Hjerppe, Roland. "Hypercatalog and Three Meta-Schemata for Database Views: Knowledge Organizing, Collection Derived, and User Established Structures." *Online Public Access to Library Files: Second National Conference*. Janet Kinsella, (editor). Elsevier International Bulletins. 101-110.

Hjerppe, Roland. "Project HYPERCATalog: Visions and Preliminary Conceptions of an Extended and Enhanced Catalog." in *Intelligent Information Systems for the Information Society*, B. C. Brookes, (editor). Amsterdam: Elsevier Science Publishers, 1986. 211-232.

Hodges, Matthew E., Ben H. Davis and Russell M. Sasnett. "Investigations in Multimedia Design Documentation." in *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*, Edward Barrett, (editor). Cambridge, MA: The MIT Press, 1989. 79-89.

Irby, Charles H. "Display Techniques for Interactive Text Manipulation." *AFIPS Conference Proceedings —1974 National Computer Conference and Exposition*. May 6-10, 1974, Chicago, IL. Montvale, NJ: AFIPS Press. 247-255.

Irish, Peggy M. and Randall H. Trigg. "Supporting Collaboration in Hypermedia: Issues and Experiences." in *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*, Edward Barrett, (editor). Cambridge, MA: The MIT Press, 1989. 90-106.

Jaffe, Conrade C., Patrick J. Lynch and Arnold W. M. Smeulders. "Hypermedia Techniques for Diagnostic Imaging Instruction: Videodisk Echocardiography Encyclopedia." *Radiology*, Vol. 117, No. 2 (May 1989): 475-80.

Jaynes, Joseph T. "Limited Freedom: Linear Reflections on Nonlinear Texts." in *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*, Edward Barrett, (editor). Cambridge, MA: The MIT Press, 1989. 148-161.

Jonassen, D. H. "Hypertext Principles for Text and Courseware Design." *Educational Psychologist*, Vol. 21 (1986): 269-292.

Jones, Henry W., III. "Developing and Distributing Hypertext Tools: Legal Inputs and Parameters." *Hypertext '87 Papers*. November 13-15, 1987, Chapel Hill, NC. New York: ACM, 1989. 367-374.

Jones, William P. "How Do We Distinguish the Hyper from the Hype in Non-linear Text?" *INTERACT '87*. H. J. and B. Shackel, (editors). September 1-4, 1987, Stuttgart. Elsevier Science Publishers B.V., 1987. 1107-1113.

Jordan, Daniel S., Daniel M. Russell, Anne-Marie S. Jensen and Russel A. Rogers. "Facilitating the Development of Representations in Hypertext with IDE." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 93-104.

Kacmar, Charles J. "A Process-Oriented Extensible Hypertext Architecture." *SIGCHI Bulletin*, Vol. 21, No. 1 (July 1989): 98-101.

Kahn, Paul. "Information Retrieval As Hypermedia: An Outline of InterBrowse." *Proceeding of the Ninth National Online Meeting*. May 10-12, New York. New York: Learned Information, 1988. 131-139.

Kahn, Paul. "Linking Together Books: Experiments in Adapting Published Material into Intermedia Documents." *Hypermedia*, Vol. 1, No. 2 (Summer 1989): 111-144.

Kahn, Paul and Norman Meyrowitz. "Guide, HyperCard, and Intermedia: A Comparison of Hypertext/Hypermedia Systems." *IRIS Technical Report*, 88-7. Brown University, Providence RI, 1988.

Kay, Alan C. "Computer Software." *Scientific American*, Vol. 251, No. 3 (September, 1984): 53-59.

Kay, Alan C. *Personal Dynamic Media*, Xerox PARC Technical Report SSL-76-1. Xerox Palo Alto Research Center, Palo Alto CA, March 1976.

Kelly, Kirk. "Online Citation Maintenance for Literature Publication and Retrieval over Computer Networks." *Teleinformatics 79*. Boutmy and Danthine, (editors). North-Holland Publishing Company, 1979.

Kerr, Elaine and Starr Roxanne Hiltz. *Computer-Mediated Communication Systems*. New York: Academic Press, 1982.

Kibby, M.R. and T. Mayes. "Towards Intelligent Hypertext." in *Hypertext: Theory into Practice*, Ray McAleese, (editor). Norwood, NJ: Ablex Publishing Corporation, 1989. 164-172.

Kochen, Manfred. "WISE: A World Information Synthesis and Encyclopedia." *Journal of Documentation*, Vol. 28 (1972): 322-343.

Koo, Richard. "A Model for Electronic Documents." *ACM SIGOIS Bulletin*. Simon Gibbs, (editor). (January 1989): 23-33.

Koved, Larry. *Restructuring Textual Information for Online Retrieval*, Technical Report 11278(#50830). IBM T.J. Watson Research Center, Yorktown Heights, NY, July 23, 1985.

Kunkel, Paul. "Hyper Media." *International Design*, (March/April 1989): 41-43.

Lambert, Steve and Suzanne Ropiequet. *CD ROM: The New Papyrus*. Redmond, WA: Microsoft Press, 1986.

Landow, George P. "Context32: Using Hypermedia to Teach Literature." *Proceedings of the 1987 IBM Academic Information Systems University AEP Conference*. Milford, Connecticut: IBM Academic Information Systems, 1987.

Landow, George P. *Course Assignments Using Hypertext: The Example of Intermedia*, IRIS, Brown University, Providence, RI, 1988.

Landow, George P. "Hypertext in Literary Education, Criticism, and Scholarship." *Computers and the Humanities*, Vol. 23 (July 1988): 173-198.

Landow, George P. "Relationally Encoded Links and the Rhetoric of Hypertext." *Hypertext '87 Papers*. November 13-15, 1987, Chapel Hill, NC. New York: ACM, 1989. 331-344.

Landow, George P. "The Rhetoric of Hypermedia: Some Rules for Authors." *Journal of Computing in Higher Education*, Vol. 1, No. 1 (Spring 1989): 39-64.

Lenat, Douglas B., Alan M. Borning, D. McDonald, C. Taylor and Stephen A. Weyer. "Knoesphere: Building Expert Systems with Encyclopedic Knowledge." *Proceedings of the 8th International Joint Conference on Artificial Intelligence*. Karlsruhe, West Germany, 1983. 167-169.

Lesk, Michael. "What to Do When There's Too Much Information." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 305-318.

Lewis, Brian T. and Jeffrey D. Hodges. "Shared Books: Collaborative Publication Management for an Office Information System." *COIS 88*. March 23-25, 1988, Palo Alto, CA. New York: ACM, 1988. 197-204.

Louie, Steven and Robert F. Rubeck. "Hypertext Publishing and the Revitalization of Knowledge." *Academic Computing*, Vol. 3, No. 9 (May 1989): 20-23, 30-31.

Lowe, David G. "SYNVIEW: The Design of a System for Cooperative Structuring of Information." *Computer-Supported Cooperative Work (CSCW '86) Proceedings*. December 3-5, Austin, TX, 1986. 376-386.

Luther, Willis and Martin Carter. *Management of Change and History in a Hypermedia Environment*, MCC Technical Report HI-164-87. June 1987.

Malone, Thomas W., Kenneth R. Grant, Franklyn A. Turbak, Stephen Brobst and Michael D. Cohen. "Intelligent Information-Sharing Systems." *Communications of the ACM*, Vol. 30, No. 5 (May 1987): 390-402.

Mantei, Marilyn and Donald L. McCracken. "Issue Analysis with ZOG, A Highly Interactive Man-Machine Interface." *Proceedings of the First International Symposium on Policy Analysis and Information Systems*, 1979.

Marchionini, Gary. "Hypermedia and Learning: Freedom and Chaos." *Educational Technology*, Vol. 27, No. 11 (1988): 8-12.

Marchionini, Gary and Ben Shneiderman. "Finding Facts and Browsing Knowledge in Hypertext Systems." *IEEE Computer*, Vol. 21, No. 1 (January 1988): 70-79.

Marshall, Catherine C. "Exploring Representation Problems Using Hypertext." *Hypertext '87 Papers*. November 13-15, 1987, Chapel Hill, NC. New York: ACM, 1989. 253-268.

Marshall, Catherine C. and Peggy Irish. "Guided Tours and On-Line Presentations: How Authors Make Existing Hypertext Intelligible for Readers." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 15-26.

Maxemchuck, Nick F. and H. A. Wilder. "Virtual Editing: I. The Concept." *Proceedings of the Second International Workshop on Office Information Systems*. October 13-15, 1982, Couvent Royal de St. Maximin. New York: Elsevier North-Holland, 1982.

McAleese, Ray. *Hypertext: Theory into Practice*. Norwood, New Jersey: Ablex Publishing Corporation, 1989.

McAleese, Ray. "Navigation and Browsing in Hypertext." in *Hypertext: Theory into Practice*, Ray McAleese, (editor). Norwood, NJ: Ablex Publishing Corporation, 1989. 6-44.

McCracken, Donald L. and Robert M. Akscyn. "Experience with the ZOG Human-Computer Interface System." *International Journal of Man-Machine Studies*, Vol. 21, No. 2 (1984): 293-310.

McCracken, Donald L. and Robert Akscyn. *The ZOG Approach to Database Management*, CS-34-113. Carnegie-Mellon University, Pittsburgh, PA.

McKnight, Cliff, John Richardson and Andrew Dillon. "The Authoring of HyperText Documents." in *Hypertext: Theory into Practice*, Ray McAleese, (editor). Norwood, NJ: Ablex Publishing Corporation, 1989. 138-147.

Meyrowitz, Norman. "Intermedia: The Architecture and Construction of an Object-Oriented Hypertext/Hypermedia System and Applications Framework." *Proceedings of the Conference on Object-Oriented Programing Systems, Languages, and Applications (OOPSLA '86)*. September 29-October 2, Portland, Oregon, 1986.

Meyrowitz, Norman. "The Missing Link: Why We're All Doing Hypertext Wrong." in *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*, Edward Barrett, (editor). Cambridge, MA: The MIT Press, 1989. 107-114.

Meyrowitz, Norman. *Networks of Scholar's Workstations: End-User Computing in a University Community*, Technical Report 85-3. IRIS, Brown University, Providence, RI, June 1985.

Michel, Stephen. "Guide — A Hypertext Solution." *CD-ROM Review*, (July/August 1987): 22-24.

Monty, Melissa L. "Temporal Context and Memory for Notes Stored in the Computer." *ACM SIGCHI Bulletin*, Vol. 18, No. 2 (October, 1986): 50-51.

Monty, Melissa L. and Thomas P. Moran. "A Longitudinal Study of Authoring Using NoteCards." *ACM SIGCHI Bulletin*, Vol. 18, No. 2 (October, 1986): 59-60.

Morariu, Janis. "Hypermedia in Instruction and Training: The Power and the Promise." *Educational Technology*, Vol. 27, No. 11 (1988): 17-20.

Moulthrop, Stuart. "Hypertext and 'the Hyperreal'." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 259-268.

Negroponte, Nicholas. "Books Without Pages." *IEEE International Conference on Communications IV*, 1979.

Negroponte, Nicholas. "An Idiosyncratic Systems Approach to Interactive Graphics." *ACM/SIGGRAPH Workshop on User-Oriented Design of Interactive Graphics Systems*. Pittsburgh, PA, October, 1976.

Nelson, Theodor H. "The Hypertext." *1965 Congress of the International Federation for Documentation (FID) Abstracts*. 10-15 October 1965, Washington DC. 80.

Nelson, Theodor H. "A File Structure for the Complex, the Changing and the Indeterminate." *Association for Computing Machinery, Proceedings of the National Conference, 20th*. New York: ACM, 1965. 84-100.

Nelson, Theodor H. "Getting it Out of Our System." in *Information Retrieval: A Critical View*, G. Schecter, (editor). Washington, D.C.: Thompson Book Co., 1967. 191-210.

Nelson, Theodor H. "As We Will Think." *Online 72: Conference Proceedings of the International Conference on Online Interactive Computing*. September, 1972, Brunel University, Uxbridge, England. Uxbridge, England: Online Computer Systems Ltd, 1973. 439-454.

Nelson, Theodor H. "A Conceptual Framework for Man-Machine Everything." *AFIPS Conference Proceedings—1973 National Computer Conference and Exposition, Proceedings*. June 4-8, 1973, New York, NY. Montvale, NJ: AFIPS Press, 1973. M21-M26.

Nelson, Theodor H. "Dream Machines: New Freedoms through Computer Screens—A Minority Report." in *Computer Lib: You Can and Must Understand Computers Now*, Redmond, WA: Microsoft Press, 1987.

Nelson, Theodor H. "Replacing the Printed Word: A Complete Literary System." in *Information Processing 80*, S.H. Lavington, (editor). North-Holland Publishing Co., IFIO 1980. 1013-1023.

Nelson, Theodor H. *Literary Machines*. Swarthmore, PA: T.H. Nelson, 1981.

Nelson, Theodor H. "A New Home for the Mind." *Datamation*, (March, 1982): 169-180.

Nelson, Theodor H. "Managing Immense Storage." *Byte*, (January 1988): 225-238.

Nelson, Theodor H. "All for One and One for All." *Hypertext '87 Papers*. November 13-15, 1987, Chapel Hill, NC. New York: ACM, 1989. v-vii.

Nelson, Theodor H. "Unifying Tomorrow's Hypermedia." *Proceedings of the 12th International Online Information Meeting*. December 6-8, 1988, London, England. Oxford and New Jersey: Learned Information, 1988. 1-7.

Neuwirth, Christine M. "Techniques of User Message Design: Developing a User Message System to Support Cooperative Work." in *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*, Edward Barrett, (editor). Cambridge, MA: The MIT Press, 1989. 325-342.

Neuwirth, Christine, David Kaufer, Rick Chimera and Terilyn Gillespie. "The Notes Program: A Hypertext Applications for Writing from Source Texts." *Hypertext '87 Papers*. November 13-15, 1987, Chapel Hill, NC. New York: ACM, 1989. 121-141.

Neuwirth, Christine M. and David S. Kaufer. "The Role of External Representation in the Writing Process: Implications for the Design of Hypertext-Based Writing Tools." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 319-342.

Newell, Alan, Donald L. McCracken, C. Kamila Robertson and Robert M. Akscyn. "ZOG and the USS CARL VINSON." *Computer Science Research Review*, (1981): 95-118.

Nguyen, L. T. and Robert A. Greenes. "A Framework for the Use of Computed Links in the EXPLORER-1 Knowledge Management System." in *MEDINFO 86, IFIP-IMIA*, R. Salamon, B. Blum and M. Jorgensen, (editors). North-Holland: Elsevier Science Publishers B.V., 1986. 891-894.

Nielsen, Jakob. "Evaluating Hypertext Usability." *Proceedings of NATO Advanced Research Workshop on Designing Hypertext/Hypermedia for Learning*. July 4-7, 1989, Rottenburg, West Germany.

Nielsen, Jakob. "Online Documentation and Reader Annotation." *Proceedings 1st International Conference on Work with Display Units*. May 12-15, 1986, Stockholm, Sweden. 526-529.

Nielsen, Jakob. "Prototyping User Interfaces Using an Object-Oriented Hypertext Programming System." *Proceedings of the NordDATA'89 Joint Scandinavian Computer Conference*. June 19-22, 1989, Copenhagen, Denmark.

Nielsen, Jakob and U. Lyngbæk. "Two Field Studies of Hypermedia Usability." *Proceedings of Hypertext 2 Conference*. June 29-30, 1989, York, UK.

Nielson, Jakob. "The Matters that Really Matter for Hypertext." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 239-248.

Nyce, James M. and Paul Kahn. "Innovation, Pragmaticism, and Technological Continuity: Vannevar Bush's Memex." *Journal of American Society for Information Science*, Vol. 40, No. 3 (May 1989): 214-220.

Oren, Tim. "The Architecture of Hypertexts." *Hypertext '87 Papers*. November 13-15, 1987, Chapel Hill, NC. New York: ACM, 1989. 291-306.

Palay, Andrew J. and Mark S. Fox. "Browsing through Databases." in *Information Retrieval Research*, R. N. Oddy, (editor). London: Butterworths, 1981. 310-324.

Parunak, H. Van Dyke. "Hypermedia Topologies and User Navigation." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 43-50.

Pasquier-Boltuck, Jacques, G. Collaud and J. Monnard. "An Object-Oriented Approach to Conceptualizing and Programming an Interactive System for the Creation and Consultation of Electronic Books." *WOODMAN '89*. May 24-31, 1989, Pennes-France.

Pasquier-Boltuck, Jacques, Edward Grossman and G. Collaud. "Prototyping an Interactive Electronic Book System Using an Object-Oriented Approach." *Proceedings of ECOOP '88*. Spring 1988.

Pearl, Amy. "Sun's Link Service: A Protocol for Open Linking." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 137-146.

Perlman, Gary. "Asynchronous Design/Evaluation Methods for Hypertext Technology Development." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 61-82.

Perry, T. S. "Hypermedia: Finally Here." *IEEE Spectrum*, Vol. 24, No. 11 (1987): 38-39.

Pontecorvo, Michael S. "Idea Processing – Concepts, Extensions and Applications." *Sperry Technology Symposium Proceedings*. May 1986, Gull Lake, MN.

Pontecorvo, Michael S. *An Idea Processing Approach to the Development of Knowledge-Based Systems*, Technical Report No. 18376. Sperry Communications Corporate Technology Center, Salt Lake City, UT, March 1986.

Pontecorvo, Michael S. and J. J. Krohnfeldt. "A Knowledge-Based Software Development Environment for the Support of Rapid Prototyping." *Univac Technology Review*, Vol. 13 (May 1987):

Potter, Richard L., Mitchell Berman and Ben Shneiderman. *An Experimental Evaluation of Three Touchscreen Strategies within a Hypertext Database*, CS-TR-2141. University of Maryland Computer Science Center, College Park, MD, November 1988.

Potts, Colin and Glenn Bruns. "Recording the Reasons for Design Decisions." *Proceedings 10th International Conference on Software Engineering*. IEEE Computer Society Press, 1988.

Price, Lynne A. "Thumb: An Interactive Tool for Accessing and Maintaining Text." *IEEE Transactions on Systems, Man, and Cybernetics*, March/April, 1982. 155-162.

Rada, Roy. "Writing and Reading Hypertext: An Overview." *Journal of American Society for Information Science*, Vol. 40, No. 3 (May 1989): 164-171.

Ragland, Craig. "Hypertext, Hypermedia, and the Macintosh." *MacA.P.P.L.E.*, (August, 1987).

Ramakrishna, K. "Schematization as an Aid to Organizing ZOG Information Nets." Computer Science Department, Carnegie-Mellon University, 1981.

Ramey, Judith. "Escher Effects in Online Text." in *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*, Edward Barrett, (editor). Cambridge, MA: The MIT Press, 1989. 388-402.

Raskin, Jef. "The Hype in Hypertext: A Critique." *Hypertext '87 Papers*. November 13-15, 1987, Chapel Hill, NC. New York: ACM, 1989. 325-330.

Raymond, Darrell R. *Personal Data Structuring in Videotex*, CS-84-7. University of Waterloo, Department of Computer Science Technology, February, 1984.

Raymond, Darrell R. and Frank Wm Tompa. "Hypertext and the Oxford English Dictionary." *Communications of the ACM*, Vol. 31, No. 7 (July 1988): 871-879.

Reitman, Walter, Bruce Roberts, Richard W. Sauvain, Daniel D. Wheeler and William Linn. "AUTONOTE - A Personal Information Storage and Retrieval System." *Proceedings of the 24th National Conference of the ACM*. August 26-28, 1969, New York: ACM, 1969. 67-75.

Remde, Joel R., Louis M. Gomez and Thomas K. Landauer. "SuperBook: An Automatic Tool for Information Exploration-Hypertext?" *Hypertext '87 Papers*. November 13-15, 1987, Chapel Hill, NC. New York: ACM, 1989. 175-188.

Robertson, C. Kamila and Robert Akscyn. "Experimental Evaluation of Tools for Teaching the ZOG Frame Editor." *Proceedings of the International Conference on Man-Machine Systems*. Manchester, UK: , July, 1982. 115-123.

Robertson, C. Kamila, Donald L. McCracken and Alan Newell. *The ZOG Approach to Man-Machine Communication*, CMU-CS-79-148. Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, October 1979.

Rubens, Philip. "Online Information, Hypermedia, and the Idea of Literacy." in *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*, Edward Barrett, (editor). Cambridge, MA: The MIT Press, 1989. 3-21.

Saffo, P. "What You Need to Know about Hypertext." *Personal Computing*, (December, 1987): 166-173.

Savoy, Jacques. "The Electronic Book EBOOKS." *The International Journal of Man-Machine Studies*, (in press).

Scacchi, Walt. "On the Power of Domain-Specific Hypertext Environments." *Journal of American Society for Information Science*, Vol. 40, No. 3 (May 1989): 183-191.

Schatz, Bruce R. *Telesophy: A System for Browsing and Sharing Inside a Large Information Space*, TM-ARH-006-094. Bell Communications Research, Morristown, NJ, September 1986.

Schatz, Bruce R. and Michael A. Caplinger. "Searching in a Hyperlibrary." *Proceedings Fifth International Conference on Data Engineering*. February 1989, Los Angeles. IEEE. 188-197.

Schnase, John L. and John J. Leggett. "Computational Hypertext in Biological Modeling." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 181-198.

Scully, John. "The Relationship between Business and Higher Education: A Perspective on the Twenty-First Century." *Educom Bulletin*, (Spring, 1988): 20-24.

Seybold, P. B. "Tymshare's Augment: Heralding a New Era." *Seybold Report on Word Processing*, Vol. 1, No. 9 (October 1978): 1-16.

Shapiro, Ezra. "A First Look at Dayflo." *Byte*, Vol. 9, No. 3 (March 1984): 81-87.

Shasha, Dennis. "NetBook—A Data Model to Support Knowledge Exploration." *Proceedings of the Eleventh International Conference on Very Large Data Bases*. Stockholm, August, 1985.

Shasha, Dennis. "When Does Non-linear Text Help?" *Proceedings of the Expert Database Systems Conference*. 1986, New York: ACM, 1986.

Shipman, Frank III, Jesse Chaney and G. Anthony Gorry. "Distributed Hypertext for Collaborative Research: The Virtual Notebook System." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 129-136.

Shneiderman, Ben. "User Interface Design and Evaluation for an Electronic Encyclopedia." *Proceedings of the 2nd International Conference on Human-Computer Interaction*. August, 1987, Honolulu, HI.

Shneiderman, Ben. "User Interface Design for the Hyperties Electronic Encyclopedia." *Hypertext '87 Papers*. November 13-15, 1987, Chapel Hill, NC. New York: ACM, 1989. 199-205.

Shneiderman, Ben, Dorothy Brethauer, Catherine Plaisant and Richard Potter. "Evaluating Three Museum Installations of a Hypertext System." *Journal of American Society for Information Science*, Vol. 40, No. 3 (May 1989): 172-182.

Shneiderman, Ben. "Reflections on Authoring, Editing, and Managing Hypertext." in *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*, Edward Barrett, (editor). Cambridge, MA: The MIT Press, 1989. 115-131.

Shneiderman, Ben and Greg Kearsley. *Hypertext Hands-On!* Reading, MA: Addison-Wesley, 1989.

Shneiderman, Ben, Philip Shafer, Roland Simon and Linda J. Weldon. *Display Strategies for Program Browsing: Concepts and an Experiment*, CAR-TR-192, CS-TR-1635. Department of Computer Science, University of Maryland, College Park, MD, 1986.

Shultz, Edward K., Roger W. Brown and J. Robert Beck. "Hypermedia in Pathology–The Dartmouth Interactive Medical Record Project." *American Journal of Clinical Pathology*, Vol. 91, No. 4, suppl. 1 (April 1989): S34-38.

Smith, John B. and Stephen F. Weiss. "Hypertext." *Communications of the ACM*, Vol. 31, No. 7 (1988): 816-619.

Smith, John B., Stephen F. Weiss and Gordon J. Ferguson. "A Hypertext Writing Environment and its Cognitive Basis." *Hypertext '87 Papers*. November 13-15, 1987, Chapel Hill, NC. New York: ACM, 1989. 195-214.

Smith, John B., Stephen F. Weiss, Gordon J. Ferguson, Jay David Bolter, M. Lansman and D. V. Beard. *WE: A Writing Environment for Professionals*, 86-025. University of North Carolina, Department of Computer Science, Chapel Hill, NC, August, 1986.

Smith, Karen E. "Hypertext–Linking to the Future." *ONLINE Magazine*, Vol. 12, No. 2 (March 1988): 32-40.

Smith, Karen E. and Stanley B. Zdonik. "Intermedia: A Case Study of the Differences Between Relational and Object-Oriented Database Systems." *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '87).* October 4-8, Orlando, FL. 16 , 1987.

Smith, Linda C. "'Memex' as an Image Potentiality in Information Retrieval Research and Development." in *Information Retrieval Research*, R. N. Oddy, (editor). London: Butterworths, 1981. 345-369.

Smolensky, Paul, Brigham Bell, Barbara Fox, Roger King and Clayton Lewis. "Constraint-based Hypertext for Argumentation." *Hypertext '87 Papers.* November 13-15, 1987, Chapel Hill, NC. New York: ACM, 1989. 215-246.

Storrs, Graham. "The Alvey DHSS Large Demonstrator Project Knowledge Analysis Tool: KANT." in *Hypertext: Theory into Practice*, Ray McAleese, (editor). Norwood, NJ: Ablex Publishing Corporation, 1989. 148-156.

Stotts, P. David and Richard Furuta. "Adding Browsing Semantics to the Hypertext Model." *Proceedings of the ACM Conference on Document Processing Systems.* December 5-9, 1988, Santa Fe, NM. New York: ACM, 1988. 43-50.

Stotts, P. David and Richard Furuta. "Petri Net Based Hypertext: Document Structure with Browsing Semantics." *ACM Transactions on Information Systems*, Vol. 7, No. 1 (January 1989): 3-29.

Streitz, Norbert A., Jörg Hanneman and Manfred Thuring. "From Ideas and Arguments to Hyperdocuments: Travelling Through Activity Spaces." *Hypertext '89 Proceedings.* November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 343-364.

Svibely, J. R. and J. W. Smith. "A Prototypic Hypertext Information System for Pathologist." *Informatics in Pathology*, Vol. 1 (1986): 133-142.

Tanguay, David A. *A General System for Managing Videotex Information Structures*, CS-86-23. University of Waterloo, Department of Computer Science Technology, June 1986.

Tchudi, S. "Invisible Thinking and the Hypertext." *English Journal*, Vol. 77, No. 1 (1988): 22-30.

Thompson, Bev and Bill Thompson. "Hyping Text: Hypertext and Knowledge Representation." *AI Expert*, (August, 1987): 25-28.

Thorsen, Linda J. and Mark Bernstein. "Developing Dynamic Documents: Special Challenges for Technical Communicators." *Proceedings of the 34th International Technical Communications Conference.* Denver, CO, 1987.

Thursh, Donald, Frank Mabry and Allan H. Levy. "Computers and Videodiscs in Pathology Education: ECLIPS as an Example of One Approach." *Human Pathlology*, Vol. 17 (1986): 216-218.

Thursh, Donald and Frank Mabry. "A Knowledge-Based Hypertext of Pathology." *Proceedings of the Fourth Annual Symposium on Computer Applications in Medical Care*, 1980.

Thursh, Donald and Frank Mabry. "A Knowledge-Based System for Pathology Education." *Bulletin of Pathology Education*, Vol. 6, No. 2 (Fall 1980): 36-45.

Thursh, Donald, Frank Mabry and Allan H. Levy. "The Knowledge Access, Management, and Extension System in Pathology." *Proceedings of the AAMSI Congress.* Allan H. Levy and B. T. Williams, (editors), 1985.

Tompa, Frank Wm. "A Data Model for Flexible Hypertext Database Systems." *ACM Transactions on Information Systems*, Vol. 7, No. 1 (January 1989): 85-100.

Tompa, Frank Wm. "Retrieving Data through Telidon." *Proceedings CIPS*, 1982.

Tompa, Frank Wm, Jan Gecsei and Gregor V. Bochmann. "Alternative Database Facilities for Videotex." in *The Telidon Book*, D. Godfrey and E. Chang, (editors). Press Porcepic, 1981.

Travers, Michael. "A Visual Representation for Knowledge Structures." *Hypertext '89 Proceedings.* November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 147-158.

Trigg, Randall H. "A Networked-based Approach to Text Handling for the On-line Scientific Community." University of Maryland, 1983.

Trigg, Randall H. "Guided Tours and Tabletops: Tools for Communicating in a Hypertext Environment." *ACM Transactions on Office Information Systems*, Vol. 6, No. 4 (October 1988): 398-414.

Trigg, Randall H. and Peggy M. Irish. "Hypertext Habitats: Experiences of Writers in NoteCards." *Hypertext '87 Papers*. November 13-15, 1987, Chapel Hill, NC. New York: ACM, 1989. 89-108.

Trigg, Randall H., Thomas P. Moran and Frank G. Halasz. "Adaptibility and Tailorability in NoteCards." *Proceedings of INTERACT '87*. September, Stuttgart, West Germany. 1987.

Trigg, Randall H. and Lucy A. Suchman. "Collaborative Writing in NoteCards." in *Hypertext: Theory into Practice*, Ray McAleese, (editor). Norwood, NJ: Ablex Publishing Corporation, 1989. 45-61.

Trigg, Randall H., Lucy A. Suchman and Frank G. Halasz. "Supporting Collaboration in NoteCards." *Computer-Supported Cooperative Work (CSCW '86) Proceedings*. December 3-5, Austin, TX, 1986. 153-162.

Trigg, Randall H. and Mark Weiser. "TEXTNET: A Network-Based Approach to Text Handling." *ACM Transactions on Office Information Systems*, Vol. 4, No. 1 (January, 1986): 1-23.

Tsai, C. "Hypertext: Technology, Applications, and Research Issues." *Journal of Educational Technology Systems*, Vol. 17, No. 1 (1988): 3-14.

Underwood, J. "Language Learning and 'Hypermedia'." *ADFL Bulletin*, Vol. 19, No. 4 (1988): 13-17.

Utting, Kenneth and Nicole Yankelovich. "Context and Orientation in Hypermedia Networks." *ACM Transactions on Information Systems*, Vol. 7, No. 1 (January, 1989): 58-84.

van Dam, Andries. *FRESS (File Retrieval and Editing System)*. Barrington, RI: Text Systems, July 1971.

van Dam, Andries. "Hypertext '87 Keynote Address." *Communications of the ACM*, Vol. 31, No. 7 (July 1988): 887-895.

van Dam, Andries and David E. Rice. "Computers and Publishing: Writing, Editing and Printing." in *Advances in Computers*, New York: Academic Press, 1970.

van Dam, Andries and David E. Rice. "On-Line Text Editing: A Survey." *Computing Surveys*, Vol. 3, No. 3 (September 1971): 93-114.

van der Merwe, D. P. "Annotating Literary Texts with Hypertext." *Proceedings of the 12th International Online Information Meeting*. December 6-8, 1988, London, England. Oxford and New Jersey: Learned Information, 1988. 239-247.

VanLehn, Kurt. *Theory Reform Caused by an Argumentation Tool*, ISL-11. Xerox Palo Alto Research Center, July, 1985.

Walker, Donald. *Knowledge Resource Tools for Accessing Large Text Files*, 85-21233-25. Bell Communications Research, 1985.

Walker, Janet H. "The Document Editor: A Support Environment for Preparing Technical Documents." *Proceedings of the ACM SIGPLAN /SIGOA Conference on Text Manipulation*. Portland, OR: , June 1981. 44-50.

Walker, Janet H. "Symbolics Sage: A Documentation Support System." *Intellectual Leverage: The Driving Technologies, IEEE Spring Compcon84*, 1984. 478-483.

Walker, Janet H. "Document Examiner: Delivery Interface for Hypertext Documents." *Hypertext '87 Papers*. November 13-15, 1987, Chapel Hill, NC. New York: ACM, 1989. 307-324.

Walker, Janet H. "The Role of Modularity in Document Authoring Systems." *Proceedings of the ACM Conference on Document Processing Systems*. December 5-9, 1988, Santa Fe, New Mexico. New York: ACM, 1988. 117-124.

Walker, Janet H. "Supporting Document Development in Concordia." *IEEE Computer*, (January, 1988): 48-59.

Walker, Janet H. "Authoring Tools for Complex Document Sets." in *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*, Edward Barrett, (editor). Cambridge, MA: The MIT Press, 1989. 132-147.

Walker, Janet H. and R. L. Bryan. "An Editor for Structured Technical Documents." *Protext IV*.

Walter, Mark. "IRIS's Intermedia: Multiuser Hypertext." *Seybold Report on Publishing Systems*, Vol. 18, No. 21 (August 7, 1989): 20-32.

Weyer, Stephen A. "As We May Learn." in *Interactive Multimedia*, Sueann Ambron and Kristina Hooper, (editors). Redmond, WA: Microsoft Press, 1988. 87-104.

Weyer, Stephen A. "The Design of a Dynamic Book for Information Search." *International Journal of Man-Machine Studies*, Vol. 17, No. 1 (July 1982): 87-107.

Weyer, Stephen A. *Searching for Information in a Dynamic Book*, Report SCG-1 (Also published as a Stanford University dissertation). Xerox Palo Alto Research Center, Palo Alto, CA, February 1982.

Weyer, Stephen A. and Alan H. Borning. "A Prototype Electronic Encyclopedia." *ACM Transactions on Office Information Systems*, Vol. 3, No. 1 (January 1985): 63-88.

White, J. E. "A High-Level Framework for Network-based Resource Sharing." *AFIPS Proceedings, National Computer Conference*. June 7-10, 1976, New York. Montvale, New Jersey: AFIPS Press, 1976. 561-570.

Wilder, H. A. and Nick F. Maxemchuck. "Virtual Editing: II. The User Interface." *Proceedings of SIGOA Conference Office Automation Systems*. June 21-23, Philadelphia, PA. New York: ACM, 1982. 41-46.

Wilson, Kathleen S. *Palenque: An Interactive Multimedia Optical Disk Prototype for Children*. Working Paper No. 2, Bank Street College of Education, Center for Children and Technology, New York, 1987.

Wilson, Kathleen S. "Palenque: An Interactive Multimedia Digital Interactive Prototype for Children." *Proceedings of the 1988 ACM Conference on Human Factors in Computer Systems (CHI '88)*, May 15-19, Washington, D.C. New York: ACM, 1988. 275-279.

Woods, William A. "What's in a Link: Foundations for Semantic Networks." in *Readings in Knowledge Representation*, Ronald J. Brachman and Hector J. Levesque, (editors). Los Altos, CA: Morgan Kaufmann, 1975.

Yankelovich, Nicole. "The Sampler Companion: Four Educational Software Samples." *Proceedings of Frontiers in Education Fifth Annual Conference*. October 19-22, Golden, CO, 1985.

Yankelovich, Nicole, L. Nancy Garrett, Karen E. Smith and Norman Meyrowitz. "Issues in Designing a Hypermedia Document System: The Intermedia Case Study." in *Interactive Multimedia*, Sueann Ambron and Kristina Hooper, (editors). Redmond, WA: Microsoft Press, 1988. 33-86.

Yankelovich, Nicole, Bernard Haan and Steven Drucker. "Connections in Context: The Intermedia System." *Proceedings of the Twenty-First Annual Hawaii International Conference on System Sciences*. Bruce D. Shriver, (editor). January 5-8, 1988, Kailua-Kona, HA. Washington, D.C.: Computer Society Press of the IEEE. 715-724.

Yankelovich, Nicole, Bernard J. Haan, Norman Meyrowitz and Steven M. Drucker. "Intermedia: The Concept and the Construction of a Seamless Information Environment." *IEEE Computer*, Vol. 21, No. 1 (January 1988): 81-96.

Yankelovich, Nicole, George Landow and Peter Heywood. *Designing Hypermedia "Ideabases"—The Intermedia Experience*, Technical Report 87-4. IRIS, Brown University, Providence, RI, 1987.

Yankelovich, Nicole, Norman Meyrowitz and Andries van Dam. "Reading and Writing the Electronic Book." *IEEE Computer*, Vol. 18, No. 10 (October 1985): 16-30.

Yankelovich, Nicole and Andries van Dam. "Spinning Scholarly Webs." *The Annenberg/CPB Project Report to Higher Education*, The Annenberg/CPB Project, Washington, D.C., 1987.

Yoder, Elise A., Robert M. Akscyn and Donald L. McCracken. "Collaboration in KMS, A Shared Hypermedia System." *Proceedings of the 1989 ACM Conference on Human Factors in Computer Systems (CHI '89)*, April 30-May 4, 1989, Austin, TX. New York: ACM, 1989. 37-42.

Yoder, Elise and Thomas C. Wettach, Esq. "Using Hypertext in a Law Firm." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 159-168.

Young, Jeffrey S. "Hypermedia." *MacWorld*, Vol. 3, No. 3 (March 1986): 116-121.

Zellweger, Polle T. "Active Paths Through Multimedia Documents." *Proceedings of the EP '88 Conference on Electronic Publishing, Document Manipulation and Typography*. April 20-22, 1988, Nice, France.

Zellweger, Polle T. "Scripted Documents: A Hypermedia Path Mechanism." *Hypertext '89 Proceedings*. November 5-7, 1989, Pittsburgh, PA. New York: ACM, 1989. 1-14.

Participants List
Hypertext Standardization Workshop

Carol A. Adams
IBM
11400 Burnet Rd.
Austin, TX 78758

Peter Aiken
George Mason University
MS ST-203
Fairfax, VA 22030-4444
paiken@gmuvax2.gmu.edu

Robert Akscyn
Knowledge Systems Inc.
4750 Old William Penn Hwy.
Murrysville, PA 15668

Frank Armour
George Mason University
MS ST-203
Fairfax, VA 22030-4444

Jean Baronas
National Institute of Standards & Technology
Room B263, Bldg. 225
Gaithersburg, MD 20899
baronas@asl.ncsl.nist.gov

Denise A. D. Bedgord
Consultant
12307 Lima Drive
Silver Spring, MD 20904

Daniel R. Benigni
National Institute of Standards & Technology
Room A266, Bldg. 225
Gaithersburg, MD 20899
benigni@ise.ncsl.nist.gov

Tim Berners-Lee
CERN
1211 Geneva 23
SWITZERLAND
tim@online.cern.ch

James D. Black
House of Representatives
MS-H2635
US House of Representatives
Washington, DC 20515
fjb@mios.house.gov

A. R. Briggs
Xerox Corporation
2000 Corp. Ridge
McLean, VA 22102

Diane Brown
Mitre Corporation
7525 Colshire Drive
Mailcode Z580
McLean, VA 22102

Karin Bruce
James Martin Associates
1850 Centennial Pk Drive
Suite 200
Reston, VA 22091

John C. Chen
Texas Instruments
P.O. Box 655474
MS 238
Dallas, TX 75265
jcen@csc.ti.com

Qi Fan Chen
Virginia Tech
Dept. of Computer Science
552 McBryde Hall
Blacksburg, VA 24061
chenq%fox@vtopus.cs.vt.edu

Paul Clapis
Hughes Danbury Optical Sy
25 Science Pk
New Haven, CT 06511
clapis@celrax.yale.cs.edu

Fred Cole
Computing Laboratory
University of Kent
Canterbury
Kent CT2 7NF
ENGLAND
fcc@ukc.ac.wk

Joe Collica
National Institute of Standards & Technology
Room A266, Bldg. 225
Gaithersburg, MD 20899
collica@ise.ncsl.nist.gov

Gregory Crane
Harvard University
Perseus Project
Dept. of Classics
319 Boylston Hall
Cambridge, MA 02138
crane@wjhl2.harvard.edu

Andrew Dove
Landmark Graphics
333 Cypress Run
Houston, TX 77094
andrew@lgc.com

Edward Edmiston
Mitre Corporation
7525 Colshire Drive
Mailcode Z580
McLean, VA 22102

Lawrence E. Fitzpatrick
Personal Library Software
15215 Shady Grove Rd
Suite 204
Rockville, MD 20850

Valerie Florance
Welch Med Library, JHU
1830 Monument Street
3rd Floor
Baltimore, MD 21205
vf@welchlab.jhu.edu

Dr. Edward A. Fox
Dept. of Computer Science
562 McBryde Hall
VPI&SU (Virginia Tech)
Blacksburg, VA 24016-0106
fox@vtopus.cs.vt.edu

David Fristrom
Interleaf
10 Canal Park
Cambridge, MA 02141

Richard Furuta
Dept. of Computer Science
University of Maryland
College Park, MD 20742
furuta@cs.umd.edu

Leonard Gallagher
National Institute of Standards & Technology
Room A266, Bldg. 225
Gaithersburg, MD 20899
gallagher@ise.ncsl.nist.gov

Kevin Gamble
USDA
3322 Smith Bldg.
Washington, DC 202500-0900
kgamble@cas.orst.edu

Bob Glushko
Search Technology, Inc
4725 Peachtree Corners Circle
Suite 200
Norcross, GA 30092
srchtec!glushko@gatech.edu

Louis Gomez
Bellcore
445 South Street
Morristown, NJ 07961
gomez@bellcore.com

Frank Halasz
Systems Sciences Laboratory
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
halasz@xerox.com

Seymour Hanfling
US Army Research Institute
5001 Eisenhower Avenue
PREI-IC
Alexandria, VA 22333

Dr. Shoshana Hardt-Kornacki
Bellcore
2A-273
445 South Street
Morristown, NJ 07961
shoshi@bellcore.com

Michael Hogan
National Institute of Standards & Technology
Room B168, Bldg. 225
Gaithersburg, MD 20899

Kris Houlahan
DEC
8300 Professional Pl
Suite 119
Landover, MD 20785

Danny B. Lange
Bruel & Kjaer Industri A/S
Department of Development
DK-2850 Naerum
DENMARK
danny.lange@bk.dk

John J. Leggett
Hypertext Research Lab
Dept. of Computer Science
Texas A&M University
College Station, TX 77843-3112
leggett@cssun.tamu.edu

William P. Loftus
Unisys Corporation
Rt. 252 & Central
1300 Wing
Paoli, PA 19301
wpl@prc.unisys.com

Kathryn C. Malcolm
Boeing Computer Corporation
P.O. Box 24346
Seattle, WA 98124-0346

Catherine Marshall
Systems Sciences Laboratory
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
marshall@xerox.com

Robert Smith Midford
Federal Computer Week
4141 N. Anderson
413
Arlington, VA 22203

Robert Miglin
ANSER Analytic Services
Crystal Gateway 3, Suite 800
1215 Jefferson Davis Hwy.
Arlington, VA 22202

Judi Moline
National Institute of Standards & Technology
Room B266, Bldg. 225
Gaithersburg, MD 20899
moline@asl.ncsl.nist.gov

Howard Moncarz
NIST
Metrology, Rm A127
Gaithersburg, MD 20899
moncarz@cme.nist.gov

Fontaine Moore
CACI, Inc.-Federal
8260 Willow Oaks Drive
Fairfax, VA 22031

Prof. Steven R. Newcomb
Center for Music Research
Florida State University
Tallahassee, FL 32306-2098
cmr!srn@bikini.cis.ufl.edu

Charles K. Nicholas
Computer Sciences Dept.
U.M.B.C.
5401 Wilkens Avenue
Catonsville, MD 21228

Dan Olson
Boeing Computer Services
P.O. Box 24346 #6498
Seattle, WA 98124

Tim Oren
Apple Computer Advanced Technology Group
Apple Computer, Inc.
20525 Mariani Ave.
MS 76-2C
Cupertino, CA 95014
oren@apple.com

Taeha Park
KAIST
P.O. Box 150
Chongryang-Dongdaeno
Seoul
KOREA
taeha@sorak.kaist.ac.kr

H. Van Dyke Parunak
Industry Technology Institute
P.O. Box 1485
Ann Arbor, MI 48106
van@iti.org

Kenneth Pugh
Information Navigation, Inc.
4201 University Drive, Suite 102
Durham, NC 27707

John J. Puttress
AT&T Bell Laboratories
600 Mountain Ave.
2C-577
Murray Hill, NJ 07974
jp@bashful.att.com

Victor Riley
IRIS/Brown University
155 George Street
Box 1946
Providence, RI 02906
var@iris.brown.edu

Louis G. Roberts
Boeing Computer Services
P.O. Box 24346
Seattle, WA 98124-0346
lroberts@atc.boeing.com

Linda Rosenberg
Goucher College
Towson, MD 20214
linda@cs.umbc.edu

Sean Sebastian
GE Info Systems
401 N. Washington St.
MC 0TCY
Rockville, MD 20850

Andrea Spinelli
Bull HN Information Systems Italia S.p.A.
Via Vittor Pisani, 10
20100 Milano
ITALY

Duane Stone
McDonnell Douglas
P.O. Box 516
MS 100 2125
St. Louis, MO 63166
stone@team-1.mdc.com

David Stotts
University of Maryland
Dept. of Computer Science
College Park, MD 20742
pds@cs.umd.edu

Craig W. Thompson
Info. Tech. Laboratory
Texas Instruments Inc.
P.O. Box 655474, MS 238
Dallas, TX 75265
thompson@csc.ti.com

Clifford Urr
Planning Analysis Corporation
Suite 890
1010 North Glebe Road
Arlington, VA 22201

Janet H. Walker, Ph.D
Digital Equipment Corp.
One Kendall Square
Bldg. 700
Cambridge, MA 02139
jwalker@crl.dec.com

David Wojick
CACI, Inc.-Federal
8260 Willow Oaks Drive
11/8
Fairfax, VA 22031

Magdalena Wright
GMA Industries
P.O. Box 16248
Arlington, VA 22215

Donald Young
McDonnell Douglas
Dept. H093/HQ
MS 100 2125
P.O. Box 516
St. Loius, MO 63166

NBS-114A (REV. 2-80)

U.S. DEPT. OF COMM.

BIBLIOGRAPHIC DATA SHEET (See instructions)

1. PUBLICATION OR REPORT NO.
NIST/SP-500/178

2. Performing Organ. Report No.

3. Publication Date
March 1990

4. TITLE AND SUBTITLE

Proceedings of the Hypertext Standardization Workshop
January 16-18, 1990, National Institute of Standards and Technology

5. AUTHOR(S)
Judi Moline, Dan Benigni, Jean Baronas

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
(formerly NATIONAL BUREAU OF STANDARDS)
U.S. DEPARTMENT OF COMMERCE
GAITHERSBURG, MD 20899

7. Contract/Grant No.

8. Type of Report & Period Covered
Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

Same as item #6

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

This report constitutes the proceedings of a three day workshop on Hypertext Standardization held at the National Institute of Standards and Technology (NIST) on January 16 - 18, 1990. Efforts towards standardization of hypertext have already been initiated in various interested organizations. In recognition of these existing efforts, NIST sponsored the Hypertext Standardization Workshop organized by the Hypertext Competence Project of the National Computer Systems Laboratory.

The major purpose of the Hypertext Standardization Workshop was to provide a forum for presentation and discussion of existing and proposed approaches to hypertext standardization. The stated workshop goals were to consider hypertext system definitions, to identify viable approaches for pursuing standards, to seek commonality among alternatives whenever possible, and to make progress towards a coordinated plan for standards development, i.e. a hypertext reference model. The workshop announcement solicitated contributed papers on any aspect of hypertext standardization, including assertions that standardization is premature or inadvisable. Approximately 30 contributions were received and distributed to the 65 workshop participants on the first day.

The workshop included plenary sessions and three discussion groups. This proceedings includes the papers selected for presentation in plenary sessions, reports of the discussion groups, and supplementary materials. Major conclusions of the workshop were that the discussion groups should continue their technical efforts, and that NIST should sponsor at least one more workshop to provide a forum for public discussion of progress.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

hypermedia; hypertext; standards

13. AVAILABILITY

☒ Unlimited
☐ For Official Distribution. Do Not Release to NTIS
☒ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.
☒ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES
259

15. Price

☆ U.S. GOVERNMENT PRINTING OFFICE: 1990--261-913/20574

USCOMM-DC 6043-P80

BEST COPY AVAILABLE

ANNOUNCEMENT OF NEW PUBLICATIONS ON COMPUTER SYSTEMS TECHNOLOGY

Superintendent of Documents
Government Printing Office
Washington, DC 20402

Dear Sir:

Please add my name to the announcement list of new publications to be issued in the series: National Institute of Standards and Technology Special Publication 500-.

Name ____

Company ____

Address ____

City ____________________ State ______________ Zip Code ____________

(Notification key N-503)

NIST Technical Publications

BEST COPY AVAILABLE

Periodical

Journal of Research of the National Institute of Standards and Technology—Reports NIST research and development in those disciplines of the physical and engineering sciences in which the Institute is active. These include physics, chemistry, engineering, mathematics, and computer sciences. Papers cover a broad range of subjects, with major emphasis on measurement methodology and the basic technology underlying standardization. Also included from time to time are survey articles on topics closely related to the Institute's technical and scientific programs. Issued six times a year.

Nonperiodicals

Monographs—Major contributions to the technical literature on various subjects related to the Institute's scientific and technical activities.

Handbooks—Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

Special Publications—Include proceedings of conferences sponsored by NIST, NIST annual reports, and other special publications appropriate to this grouping such as wall charts, pocket cards, and bibliographies.

Applied Mathematics Series—Mathematical tables, manuals, and studies of special interest to physicists, engineers, chemists, biologists, mathematicians, computer programmers, and others engaged in scientific and technical work.

National Standard Reference Data Series—Provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated. Developed under a worldwide program coordinated by NIST under the authority of the National Standard Data Act (Public Law 90-396). NOTE: The Journal of Physical and Chemical Reference Data (JPCRD) is published quarterly for NIST by the American Chemical Society (ACS) and the American Institute of Physics (AIP). Subscriptions, reprints, and supplements are available from ACS, 1155 Sixteenth St., NW., Washington, DC 20056.

Building Science Series—Disseminates technical information developed at the Institute on building materials, components, systems, and whole structures. The series presents research results, test methods, and performance criteria related to the structural and environmental functions and the durability and safety characteristics of building elements and systems.

Technical Notes—Studies or reports which are complete in themselves but restrictive in their treatment of a subject. Analogous to monographs but not so comprehensive in scope or definitive in treatment of the subject area. Often serve as a vehicle for final reports of work performed at NIST under the sponsorship of other government agencies.

Voluntary Product Standards—Developed under procedures published by the Department of Commerce in Part 10, Title 15, of the Code of Federal Regulations. The standards establish nationally recognized requirements for products, and provide all concerned interests with a basis for common understanding of the characteristics of the products. NIST administers this program as a supplement to the activities of the private sector standardizing organizations.

Consumer Information Series—Practical information, based on NIST research and experience, covering areas of interest to the consumer. Easily understandable language and illustrations provide useful background knowledge for shopping in today's technological marketplace.

Order the **above** *NIST publications from: Superintendent of Documents, Government Printing Office, Washington, DC 20402.*

Order the **following** *NIST publications—FIPS and NISTIRs—from the National Technical Information Service, Springfield, VA 22161.*

Federal Information Processing Standards Publications (FIPS PUB)—Publications in this series collectively constitute the Federal Information Processing Standards Register. The Register serves as the official source of information in the Federal Government regarding standards issued by NIST pursuant to the Federal Property and Administrative Services Act of 1949 as amended, Public Law 89-306 (79 Stat. 1127), and as implemented by Executive Order 11717 (38 FR 12315, dated May 11, 1973) and Part 6 of Title 15 CFR (Code of Federal Regulations).

NIST Interagency Reports (NISTIR)—A special series of interim or final reports on work performed by NIST for outside sponsors (both government and non-government). In general, initial distribution is handled by the sponsor; public distribution is by the National Technical Information Service, Springfield, VA 22161, in paper copy or microfiche form.

BEST COPY AVAILABLE